Lecture Notes in Computer Science 4835

Commenced Publication in 1973
Founding and Former Series Editors:
Gerhard Goos, Juris Hartmanis, and Jan van Leeuwen

Takeshi Tokuyama (Ed.)

Algorithms and Computation

18th International Symposium, ISAAC 2007
Sendai, Japan, December 17-19, 2007
Proceedings

Springer

Volume Editor

Takeshi Tokuyama
Tohoku University
Graduate School of Information Sciences
Sendai, 980-8579, Japan
E-mail: tokuyama@dais.is.tohoku.ac.jp

Library of Congress Control Number: 2007940402

CR Subject Classification (1998): F.2, C.2, G.2-3, I.3.5, C.2.4, E.5

LNCS Sublibrary: SL 1 – Theoretical Computer Science and General Issues

ISSN 0302-9743
ISBN-10 3-540-77118-2 Springer Berlin Heidelberg New York
ISBN-13 978-3-540-77118-0 Springer Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 12200528 06/3180 5 4 3 2 1 0

Preface

ISAAC 2007, the 18th International Symposium on Algorithms and Computation took place in Sendai, Japan, December 17-19, 2007. In the past, it was held in Tokyo (1990), Taipei (1991), Nagoya (1992), Hong Kong (1993), Beijing (1994), Cairns (1995), Osaka (1996), Singapore (1997), Daejeon (1998), Chennai (1999), Taipei (2000), Christchurch (2001), Vancouver (2002), Kyoto (2003), Hong Kong (2004), Hainan (2005), and Kolkata(2006).

The symposium provided a forum for researchers working in algorithms and the theory of computation from all over the world. In response to our call for papers we received 220 submissions from 40 countries. The task of selecting the papers in this volume was done by our Program Committee and many other external reviewers. After a thorough review process, the Committee selected 77 papers. We hope all accepted papers will eventually appear in scientific journals in a more polished form. Two special issues, one of [illegible] and one of the [illegible], with selected papers from ISAAC 2007 are in preparation.

The best paper award was given for "Integer Representation and Counting in the Bit Probe Model" to Mohammad Rhaman and Ian Munro. Selected from 27 submissions authored by only students, the best student paper awards were given for "On Mixing and Edge Expansion Properties in Randomized Broadcasting" to Thomas Sauerwald and for "Faster Combinatorial Algorithms for Determinant and Pfaffian" to Anna Urbanska. Two eminent invited speakers, Pankaj K. Agarwal, Duke University, USA, and Robin Thomas, Georgia Institute of Technology, USA, also contributed to this volume.

I would like to thank the Program Committee members and many external reviewers for their great efforts in the review process. I would also like to thank the Conference Committee, Local Organizing Committee, and ISAAC Advisory Committee for their contribution to make the conference a success. Finally, I would like to thank our sponsors and supporting organizations for their assistance and support.

December 2007 Takeshi Tokuyama

Organization

ISAAC2007 was jointly organized by the Graduate School of Information Sciences and Research Institute of Electrical Communication of Tohoku University.

Program Committee

Takao Asano (Chuo University, Japan)
Prosenjit Bose (Carleton University, Canada)
Jin-Yi Cai (Wisconsin University, USA)
Timothy Chan (University of Waterloo, Canada)
Chandra Chekuri (UIUC, USA)
Danny Z. Chen (University of Notre Dame, USA)
Siu-Wing Cheng (HKUST, Hong Kong)
Otfried Cheong (KAIST, Korea)
Satoshi Fujita (Hiroshima University, Japan)
Magnus Halldorsson (University of Iceland, Iceland)
Seok-Hee Hong (University of Sydney, Australia)
Peter Hoyer (University of Calgary, Canada)
Xiaodong Hu (Chinese Academy of Science, China)
Satoru Iwata (Kyoto University, Japan)
Piotr Krysta (University of Liverpool, UK)
Der-Tsai Lee (Academia Sinica, Taiwan)
Xuemin Lin (University of New South Wales, Australia)
Ming Li (University of Waterloo, Canada)
Toshimitsu Masuzawa (Osaka University, Japan)
Keiji Matsumoto (NII, Japan)
David Mount (University of Maryland, USA)
Seffi Naor (Technion, Israel and Microsoft Research)
Pandu Rangan (IIT, India)
Kunihiko Sadakane (Kyushu University, Japan)
Baruch Schieber (IBM Research, USA)
Jun Tarui (UEC, Japan)
Takeshi Tokuyama (Tohoku University, Japan) (Chair)
Dorothea Wagner (University of Karlsruhe, Germany)
Frances Yao (City University, Hong Kong)
Xiao Zhou (Tohoku University, Japan)

Organizing Committee

Takao Nishizeki (Tohoku University, Japan)(Symposium Chair)
Ayumi Shinohara (Tohoku University, Japan)(Organizing Co-chair)
Xiao Zhou (Tohoku University, Japan)(Organizing Co-chair)
Jinhee Chun (Tohoku University, Japan)
Takehiro Ito (Tohoku University, Japan)
Akiyoshi Shioura (Tohoku University, Japan)

ISAAC Advisory Committee

Francis Chin (University of Hong Kong, China)
Ding-Zhu Du (University of Texas at Dallas, USA)
Peter Eades (University of Sydney and NICTA, Australia)
Wen-Lian Hsu (Academia Sinica, Taiwan)
Toshihide Ibaraki (Kwansei Gakuin University, Japan)
Der-Tsai Lee (Academia Sinica, Taiwan)
Takao Nishizeki (Tohoku University, Japan, Chair)

Sponsors

Graduate School of Information Sciences, Tohoku University
Research Institute of Electrical Communication, Tohoku University
New Horizons in Computing, Scientific Research on Priority Areas
The Telecommunications Advancement Foundation
Kayamori Foundation of Informational Science Advancement
Support Center for Advanced Telecommunications Technology Research
Sendai Tourism and Convention Bureau

External Referees

Eric Allender	Kazuyuki Amano	Elliot Anshelevich
Lars Arge	Boris Aronov	Yasuhito Asano
Moshe Babaioff	Sang Won Bae	Nikhil Bansal
Jeremy Barbay	Reinhard Bauer	Michael Baur
Glencora Borradaile	Peter Brass	Patrick Briest
Gerth Brodal	Kevin Buchin	Maike Buchin
Paz Carmi	Erin Chambers	Richard Chang
Michael Charleston	Ke Chen	Eric Chen
Xi Chen	Julia Chuzhoy	Graham Cormode
Artur Czumaj	Ovidiu Daescu	Peter Damaschke
Mirela Damian	Daniel Delling	Xiaotie Deng

Guoli Ding
Friedrich Eisenbrand
Sándor Fekete
Pierre Fraigniaud
Marco Gaertler
Xavier Goaoc
Torben Hagerup
Martin Held
Martin Holzer
Tsan-sheng Hsu
Takehiro Ito
Chuzo Iwamoto
Jesper Jansson
Daniel Johannsen
Mong-Jen Kao
Bastian Katz
Sanjeev Khanna
Christian Knauer
Jochen Konemann
Dieter Kratsch
Yoshiyuki Kusakari
Juergen Lerner
Jian Li
Katrina Ligett
Tien-Ching Lin
Pinyan Lu
Meena Mahajan
Daniel Marx
Brendan McKay
Sascha Meinert
Kazuyuki Miura
Hiroshi Nagamochi
Gen Nishikawa
Martin Nöllenburg
Hirotaka Ono
Evanthia Papadopoulou
Sheung-Hung Poon
San Ratanasanya
Iris Reinbacher
Jared Saia
Thomas Schank
Ayumi Shinohara
Christian Sohler
Tomoko Suzuki
Suguru Tamaki

Catalin Dohotaru
Thomas Erlebach
Abraham Flaxman
Akihiro Fujiwara
Rajiv Gandhi
Robert Görke
Sariel Har-Peled
Harald Hempel
Samuel Hornus
Piotr Indyk
Tsuyoshi Ito
Taisuke Izumi
Bin Jiang
Hirotsugu Kakugawa
Yoshiaki Katayama
Ken-ichi Kawarabayashi
Samir Khuller
Hirotada Kobayashi
Guy Kortsarz
Piyush Kumar
Troy Lee
Christos Levcopoulos
Cheng-Chung Li
Hyeong-Seok Lim
Xiaomin Liu
Jun Luo
Anil Maheshwari
Claire Mathieu
Frank McSherry
Matúš Mihal'ák
Yuichiro Miyamoto
Shin-ichi Nakano
Harumichi Nishimura
Yoshio Okamoto
Fukuhito Ooshita
Kunsoo Park
Jaikumar Radhakrishnan
Dror Rawitz
Ignaz Rutter
Mohammad Salavatipour
Bernhard Scholz
Akiyoshi Shioura
Aravind Srinivasan
Maxim Sviridenko
Tami Tamir

Christoph Durr
Lene Favrholdt
Paola Flocchini
Stefan Funke
Sumit Ganguly
Sudipto Guha
Xin He
Martin Hoefer
Anchen Hsiao
Toshimasa Ishii
Hiro Ito
Riko Jacob
Minghui Jiang
Iyad Kanj
Matthew J. Katz
Neeraj Kayal
Shuji Kijima
Satoshi Kobayashi
Dariusz Kowalski
Satoru Kuroda
Francois Legall
Liane Lewin-Eytan
Chung-Shou Liao
Ching-Chi Lin
Hsueh-I Lu
Yi Luo
Barnaby Martin
Andrew McGregor
Steffen Mecke
Vahab Mirrokni
Hiroki Morizumi
Giri Narasimhan
Zeev Nutov
Tatsuaki Okamoto
James Palmer
Georgios Piliouras
Rajeev Raman
Rudy Raymond
Bardia Sadri
Nicolas Schabanel
Roy Schwartz
Michiel Smid
Xiaomin Sun
Grzegorz Swirszcz
Chuan Yi Tang

Kazushige Terui
Sebastiaan Terwijn
Thomas Thierauf
Seinosuke Toda
Etsuji Tomita
Chun-Hung Tsai
Tatsuhiro Tsuchiya
Helen Tu
Ryuhei Uehara
Takeaki Uno
Rob Van Stee
Kasturi Varadarajan
Gert Vegter
Anastasios Viglas
Antoine Vigneron
Koichi Wada
Lusheng Wang
Yajun Wang
Wei Wang
Hom-Kai Wang
Hoeteck Wee
Thomas Wexler
Carsten Witt
Nicholas Wormald
Yu Wu
Xiaodong Wu
Dachuan Xu
Jinhui Xu
Mutsunori Yagiura
Atsuko Yamaguchi
Yukiko Yamauchi
Koichi Yamazaki
Mark Yim
Teng-Kai Yu
Hamid Zarrabi-Zadeh
Wenjie Zhang
Ying Zhang
Binhai Zhu
Uri Zwick

Table of Contents

2B Graph Drawing

3A Distributed Algorithms

3B Optimization I

4A Data Structure I

4B Game Theory

5A Database Applications

5B Online Algorithms

6A I/O Algorithms

6B Networks

7A Optimization II

7B Computational Geometry II

8A Geometric Applications

8B Data Structures II

9A Computational Geometry III

9B Complexity II

10A String

10B Graph Algorithms II

Modeling and Analyzing Massive Terrain Data Sets

Pankaj K. Agarwal

Department of Computer Science,
Duke University,
Durham NC 27708-0129, USA
pankaj@cs.duke.edu

With recent advances in terrain-mapping technologies such as Laser altimetry (LIDAR) and ground based laser scanning, millions of georeferenced points can be acquired within short periods of time. However, while acquiring and georeferencing the data has become extremely efficient, transforming the resulting massive amounts of heterogeneous data to useful information for different types of users and applications is lagging behind, in large part because of the scarcity of robust, efficient algorithms for terrain modeling and analysis that can handle massive data sets acquired by different technologies and that can rapidly detect and predict changes in the model as the new data is acquired.

This talk will review our on-going work on developing efficient algorithms for terrain modeling and analysis that work with massive data sets. It will focus on algorithms for constructing digital elevation models of terrains, handling noise in elevation models, and for computing watershed regions and stream-networks. The talk will also discuss some of the challenges that we face in this area.

T. Tokuyama (Ed.): ISAAC 2007, LNCS 4835, p. 1, 2007.

Coloring Triangle-Free Graphs on Surfaces

Zdeněk Dvořák[1], Daniel Král'[2], and Robin Thomas[3,⋆]

[1] Institute for Theoretical Computer Science (ITI), Charles University, Malostranské náměstí 25, 118 00 Prague, Czech Republic
kral@kam.mff.cuni.cz
[2] Department of Applied Mathematics, Charles University, Malostranské náměstí 25, 118 00 Prague, Czech Republic
rakdver@atrey.karlin.mff.cuni.cz
[3] School of Mathematics, Georgia Institute of Technology, Atlanta, GA 30332-0160, USA
thomas@math.gatech.edu

1 Introduction

We are concerned with coloring graphs that embed in a fixed surface. This restriction often makes coloring problems tractable, and the purpose of this abstract is to describe a new result along these lines.

By a [illegible] we mean a compact 2-dimensional manifold with empty boundary. The classification theorem of surfaces states that every surface is homeomorphic to either the surface S_g obtained from the sphere by adding g handles ("the orientable surface of genus g"), or to the surface N_k obtained from the sphere by adding k "cross-caps" ("the non-orientable surface of cross-cap number k"). We refer to [8] for background information on surfaces and graphs embedded in them.

Thomassen initiated a systematic study of coloring graphs on surfaces by formulating two related questions. The first one asks for the complexity status of the following decision problem for fixed integers k and g and a fixed surface Σ.

Decision Problem 1. **k-COLORING GIRTH $\geq q$ GRAPHS IN Σ**
[illegible] G [illegible] q [illegible] Σ
[illegible] G k [illegible]

The second question is closely related and concerns the finiteness of $(k+1)$-critical graphs among the instances of Decision Problem 1. A graph G is $(k+1)$ [illegible] if it is not k-colorable, but every proper subgraph is k-colorable. The second question Thomassen posed is

[illegible] Given integers k, q and a surface Σ, is the number of $(k+1)$-critical graphs of girth at least q that embed into Σ finite?

If the answer is yes, then Decision Problem 1 has an easy polynomial-time algorithm—simply test if the input graph G has a subgraph isomorphic to one of the finitely many $(k+1)$-critical graphs. In fact, this algorithm can be implemented to run in linear time [2]. Furthermore, in all instances when the number

⋆ Partially supported by NSF Grants No. DMS-0200595 and DMS-0354742.

T. Tokuyama (Ed.): ISAAC 2007, LNCS 4835, pp. 2–4, 2007.

of $(k+1)$-critical graphs is finite we also have an explicit bound on the maximum size of such a graph, and hence we not only know that the algorithm exists, but we can actually construct it.

Question 1 is now completely settled. For $k = 2$ the answer is negative, because of odd cycles, and for $k = 3$ and $q \leq 4$ the answer is negative for all surfaces other than the sphere because of the graphs obtained from odd cycles by means of the Mycielski's construction [1, Section 8.5]. For $k = 4$, $q = 3$ the answer is negative because of a construction of Fisk [3]. For pairs k, q such that $q \geq 6$ and $k \geq 3$, or $q \geq 4$ and $k \geq 4$, or $k \geq 7$ the answer is positive by Euler's formula and a theorem of Gallai [4]; see [6]. In the remaining cases $k = 5$, $q = 3$ and $k = 3$, $q = 5$ the answer is positive by two deep theorems of Thomassen [9,10]. Let us state the one directly related to our work.

Theorem 2. [illegible]

It follows from the solution to Question 1 that Decision Problem 1 is polynomial-time solvable for all pairs (k, q), except the pairs $(3, 3)$, $(3, 4)$ and $(4, 3)$. In case of the last pair Problem 1 has a trivial solution when Σ is the sphere, but for all other surfaces its resolution is clouded by similar issues that make the Four-Color Problem so difficult. Thus the prospects for a solution are not very bright. In case of the pair $(3, 3)$ Problem 1 is NP-hard for all surfaces, because it is NP-hard even when Σ is the sphere [5, Theorem 4.2]. That leaves us with the case $k = 3$ and $q = 4$, which constitutes our main result.

2 Our Result

The classical theorem of Grötzsch [7] states that every triangle-free planar graph is 3-colorable. Thus deciding whether a triangle-free planar graph is 3-colorable is trivial. However, Grötzsch's Theorem does not generalize to any other surface Σ, and, in fact, the 3-colorability of triangle-free graphs embedded in Σ is an interesting problem. When Σ is the projective plane it has been solved by Gimbel and Thomassen [6], but it was open for all other surfaces. In this abstract we announce solution for all surfaces, as follows.

Theorem 3. [illegible]

In fact, our result is more general in two respects. The input graph is allowed to have triangles, as long as they are not trivial. (We say that a cycle C is [illegible], if it bounds a disk, and non-trivial otherwise.) Second, a bounded number of vertices may be precolored.

We prove Theorem 3 by means of the following structural result. We need a definition first. If G is a graph embedded in a surface Σ and f is a face of a subgraph of G, then by $G[f]$ we denote the subgraph of G consisting of all vertices

and edges of G embedded in the closure of f. Furthermore, we regard $G[f]$ as embedded in the surface $\Sigma[f]$ obtained from f by capping off each component of the boundary of f by a disk.

Theorem 4. [illegible] Σ [illegible] N [illegible] G [illegible] Σ [illegible] 3 [illegible] H [illegible] N [illegible] f [illegible] H

(i) [illegible]

(ii) [illegible]

(iii) [illegible] $G[f]$ [illegible] $\Sigma[f]$ [illegible]

We omit the definition of local planarity. Instead, let us remark that we have proven a coloring extension theorem that under the assumption of local planarity gives an easily checkable necessary and sufficient condition for a coloring of the boundary of f to extend to $G[f]$. This condition is in terms of "winding number" of the precoloring, and takes a different form depending on whether $\Sigma[f]$ is orientable or not.

The proof of Theorem 4 can be converted to a polynomial-time algorithm to find the graph H. Starting with the null graph, if the current graph does not satisfy any of the conclusions of the theorem, then we find a way to enlarge H while simplifying the faces of H, and repeat. The simplification guarantees that there will be only a constant number of iterations.

Once the graph H is found we use Theorem 4 to test whether some 3-coloring of H extends to $G[f]$ for all faces f of H. We have developed separate algorithms to do that when f is a disk or a cylinder. If condition (iii) holds, then the extension question can be easily decided using the coloring extension theorem mentioned above.

References

1. Bondy, J.A., Murty, U.S.R.: Graph Theory with Applications. North-Holland, New York, Amsterdam, Oxford (1976)
2. Eppstein, D.: Subgraph isomorphism in planar graphs and related problems. J. Algor. and Appl. 3, 1–27 (1999)
3. Fisk, S.: The nonexistence of colorings. J. Combin Theory Ser. B 24, 247–248 (1978)
4. Gallai, T., Graphen I, II, K.: Publ. Math. Inst. Hungar. Acad. Sci. 8, 165–192 and 373–395 (1963)
5. Garey, M.R., Johnson, D.S.: Computers and intractability. A guide to the theory of NP-completeness. W. H. Freeman, San Francisco (1979)
6. Gimbel, J., Thomassen, C.: Coloring graphs with fixed genus and girth. Trans. Amer. Math. Soc. 349, 4555–4564 (1997)
7. Grötzsch, H., Dreifarbensatz für dreikreisfreie Netze auf der Kugel, E., Wiss, Z.: Martin-Luther-Univ. Halle-Wittenberg Math.-Natur. Reihe. 8, 109–120 (1959)
8. Mohar, B., Thomassen, C.: Graphs on surfaces. Johns Hopkins University Press, Baltimore, MD (2001)
9. Thomassen, C.: Color-critical graphs on a fixed surface. J. Combin. Theory Ser. B 70, 67–100 (1997)
10. Thomassen, C.: The chromatic number of a graph of girth 5 on a fixed surface. J. Combin. Theory Ser. B 87, 38–71 (2003)

Integer Representation and Counting in the Bit Probe Model*

M. Ziaur Rahman and J. Ian Munro

Cheriton School of Computer Science, University of Waterloo,
200 University Ave W, Waterloo, Ontario, Canada N2L 3G1
mzrahman@cs.uwaterloo.ca, imunro@uwaterloo.ca

Abstract. We examine the problem of integer representation in near minimal number of bits so that increment and decrement (and indeed addition and subtraction) can be performed using few bit inspections and fewer bit changes. In particular, we prove a new lower bound of $\Omega(\sqrt{n})$ for the increment and decrement operation, where n is the minimum number of bits required to represent the number. The model of computation we considered is the bit probe model, where the complexity measure counts only the bitwise accesses to the data structure. We present several efficient data structures to represent integer that use a logarithmic number of bit inspections and a constant number of bit changes per operation.

Keywords: bit probe model, data structure, Gray code, lower bound.

1 Introduction

The data type integer is fundamental to any computer and any programming language. Therefore, the representation of integers and operations on them are fundamental issues. We study the problem of integer representation using a nearly minimal number of bits so that basic operations can be done efficiently. The operations include increment, decrement, addition and subtraction.

The model of computation considered in this paper is the bit probe model. In this model, the complexity measure counts the bitwise accesses to the data structure. The model ignores the cost of computation. Lower bounds derived by this model are also lower bounds in any realistic, sequential model of computation.

Although the cell probe model is used more frequently in the data structures, the bit probe complexity measure was never out of sight and has gained interest in recent years [2,12,15,16]. The bit probe model was introduced by Minsky and Papert [13] and was generalized into the cell probe model by Yao later [17].

2 Preliminaries

We are interested in representing the integers in the range $[0, 2^n - 1]$, where n is the minimum number of bits required to represent an element of this set, and we

* This work was supported by NSERC of Canada and The Canada Research Chairs Program.

T. Tokuyama (Ed.): ISAAC 2007, LNCS 4835, pp. 5–16, 2007.

call n the dimension of an integer in this range. Let x be an integer of dimension n. The increment operation yields a representation of the value $(x+1) \bmod 2^n$. Similarly, the decrement operation can be defined. The addition operation has the form $x \leftarrow x+y$, where x has a larger dimension than y, so that the number x is replaced with the sum. The subtraction operation can be defined similarly.

The standard binary number system uses n bits to represent an integer of dimension n. To increment, scan the bits from the right until the rightmost '0'. Flip this bit to '1' and all previous bits to '0's. The decrement operation is similar. So, this representation requires n bit probes per operation in the worst case. The addition between x and y proceeds from the rightmost bits of x and y. Add the corresponding bits with the proper propagation of carry. Clearly, the operation requires $n+m$ bit inspections and $n+1$ bit changes in the worst case.

The redundant binary system supports constant amortized time per operation [3,4]. The scheme doubles the space usage. But the model considered in these schemes is not the bit probe model. Frandsen et al. [8] gave a representation that requires $O(\log n)$ bit probes per operation. Using the same time, the number can be tested to be equal to a number from chosen a fixed set of numbers. Fredman used the decision assignment tree to obtain the bit probe lower bound of $\Omega(\log n)$ to generate a quasi-Gray code sequence [9]. Fredman's tree representation uses three times the minimum space required while permitting a constant number of bit changes for the increment operation.

The Binary Reflected Gray Code (BRGC) was invented by Frank Gray while converting analog signals into digital signals [10]. A Gray code sequence of dimension n contains a sequence of 2^n elements, where each element of the sequence is a n-bit code, such that two consecutive codes in the sequence differ only in one bit. In cyclic Gray code, the first and the last element in the sequence also differ in one bit. The BRGC is a cyclic Gray code.

Definition 1. *The empty string is the sequence $G(0)$, the Binary Reflected Gray Code (BRGC) sequence of dimension 0. Let $G(n-1)$ be the BRGC sequence of dimension $n-1$. Then the following sequence: $G(n) = 0.[G(n-1)], 1.[G(n-1)]^R$ is the BRGC sequence of dimension n, where $[G(n-1)]^R$ is the BRGC sequence of dimension $n-1$ in the reverse order. The '.' indicates concatenation with each code of the sequence and ',' separates the first half of the sequence from the last half.*

Let X be the BRGC of dimension n that contains a value $x \in [0, 2^n - 1]$ and is denoted by $X = x_n x_{n-1} \cdots x_1$, where $x_i \in [0, 1]$. The definition of the BRGC sequence leads to the following observations.

Observation: In the BRGC sequence of dimension n, the i-th bit is flipped 2^{n-i} times, for all i such that $1 \leq i < n$. The n-th bit is flipped 2 times.

Observation: In the BRGC sequence of dimension n, a transition from a code in the sequence to the next code causes a bit flip. Consider the flip of bit x_i. (i) In each of the next $2^{i-1} - 1$ transitions, a bit x_j is flipped, where $i > j$. (ii) A bit x_k is flipped in the 2^{i-1}-th transition, where $i < k$. (iii) The distance between two flips of bit x_i is 2^i.

The generation of the BRGC has been studied extensively. To get the next code, for an even parity code, flip the rightmost bit. For an odd parity code, find the rightmost '1' and flip the bit to its left. The only special case is when the last bit x_n is the only '1' in the code. This is also the last code in the BRGC sequence. Note that the parity is odd. But in this case, it is sufficient to flip bit x_n to go back to the first code of the BRGC. Boothroyd [1] calculates the parity and finds the last '1' bit by a scan of the bits. With no additional space, the algorithm to generate the next code of the BRGC sequence requires n bits inspections, as it has to determine the parity of the code. But only one bit needs to be changed to get the next code. Misra [14] stores the parity explicitly and maintains a stack of indices of bits that are '1's to improve the performance significantly on average. Er [6] presented some improvements on Misra's algorithm. None of the algorithms mentioned above, considers the problem in the bit probe model. Lucal proposed a modified Gray code where the parity bit is integrated into the code [11]. For detailed survey on the Gray code, the readers are referred to the paper by Doran [5].

3 A Lower Bound

In this section, we present a new lower bound for the increment operation of an integer with dimension n. The Sunflower lemma is used to prove the lower bound. A Sunflower with p petals is a collection $S_1, S_2, \cdots, S_p$ of sets so that the intersection $S_i \cup S_j$ is the same for each pair of distinct indices i and j. The intersection is called the center of the sunflower. The following well-known lemma is due to Erdös and Rado [7].

Lemma 1. *(Sunflower Lemma) Let $S_1, S_2, \cdots, S_m$ be a system of sets each of cardinality at most l. If $m > (p-1)^{l+1}l!$, then the collection contains as a subcollection a sunflower with p petals.*

Let M be a memory of size n bits. There can be $m = 2^n$ distinct memory configurations. A counting sequence $s = c_0c_1 \cdots c_{m-1}c_0$ is a sequence of distinct memory configurations such that for any two memory configurations c_i and c_j, if $i \neq j$ then $c_i \neq c_j$. The increment algorithm is denoted by a Decision Assignment Tree (DAT) where at each internal node a single bit is inspected and at each leaf one or more bits are flipped, so that the algorithm is ignorant of the current memory configuration.

Theorem 1. *Consider an integer of dimension n represented in exactly n bits using any representation of the values in the range. The increment operation on this representation requires $\Omega(\sqrt{n})$ bit inspections in the worst case.*

Proof. Fix a counting sequence $s = c_0c_1 \cdots c_{m-1}c_0$. Let S_i be the set of bits inspected while switching from configuration c_i to configuration $c_{(i+1) \bmod m}$, for all i such that $m > i \geq 0$. Build a DAT for the sequence s, where at each internal node a single bit is inspected and at each leaf one or more bits are

flipped. For a memory configuration c_i, follow the root-to-leaf path and flip the bits mentioned at the leaf to get the memory configuration c_{i+1}. Let l be the height of the DAT. Therefore, $|S_i| \leq l$, for all i such that $m > i \geq 0$.

Let p' be the largest integer satisfying the inequality in the sunflower lemma. Therefore, we can find a sunflower $S_{i_1} \cdots S_{i_{p'}}$ with p' petals for all j, such that $0 \leq i_j < m$ and $1 \leq j \leq p'$. Let C be the center of the sunflower. Thus C is a set of bits. Since p' is the largest integer that satisfies the inequality, we have, $m \leq \{(p'+1)-1\}^{l+1} l!$ and hence $\log p' \geq \frac{\log m}{l+1} - O(\log l)$.

Consider two sets S_i and S_j such that they correspond to two petals of the sunflower. The sets S_i and S_j correspond to two root-to-leaf paths in the DAT. Consider the bit inspected at the lowest common node of these two paths. The bit is in C as it is a common bit of S_i and S_j. Furthermore, the value of the bit must differ for S_i and S_j. Therefore, the content of C must uniquely identify a petal of the sunflower. Thus, $|C| \geq \log p' \geq \frac{\log m}{l+1} - O(\log l)$. But, we know $|C| \leq l$. Therefore, $l \geq \frac{\log m}{l+1} - O(\log l) \Rightarrow l = \Omega(\sqrt{\log m})$. Since $m = 2^n$, we have $l = \Omega(\sqrt{n})$. □

4 Efficient Increment and Decrement

Our first data structure for efficient integer representation uses the properties of the BRGC and requires $n + \log n + 3$ bits to represent an integer of dimension n. Store the BRGC of dimension n explicitly. Also store a parity bit for the Gray code — the parity bit is '1' when the current code has even number of '1's. The n bits of the code are divided into two blocks — (i) the lower order $\log n$ bits of the code denoted by $X_L = x_{\log n} \cdots x_1$, and (ii) the remaining $(n - \log n)$ bits of the code denoted by $X_H = x_n \cdots x_{\log n + 1}$. A pointer P, requiring $\log n$ bits, points to the rightmost '1' in X_H, when it is ready. The pointer P becomes invalid, when the position of the rightmost '1' of X_H is changed. Update P in the background by erasing the previous content of P and by checking one bit of X_H at a time. We use two status bits — (i) R : the ready bit, denotes whether the pointer P is pointing to the rightmost '1' of X_H, and (ii) E : the erase bit, denotes whether P is being initialized by erasing the previous content.

Lemma 2. *The position of the rightmost '1' of X_H is changed on an increment or decrement operation only when the rightmost bit of X_H is flipped, except for the special case, where bit x_n is flipped from '1' to '0' ('0' to '1') in an increment (decrement) operation.*

Proof. Let x_i be the bit of X_H to be flipped next by an increment (decrement) operation. We have, $i > \log n$. From the increment (decrement) algorithm of the BRGC, we know that, x_i is flipped only when the parity of the code is odd (even) and x_{i-1} is the rightmost '1' in the code. If $i > \log n + 1$, then x_{i-1} remains the rightmost '1' of X_H. Now, consider $i = \log n + 1$. If the bit x_i is flipped from '0' to '1' then, it becomes the new rightmost '1' of X_H. Let x_i is flipped from '1' to '0'. Before the flip, x_i was the rightmost '1' of X_H. As the bit is changed to '0', the rightmost '1' of X_H is also changed. □

A valid pointer P contains the distance of the rightmost '1' of X_H from $x_{\log n+1}$, the rightmost bit of X_H. If the rightmost bit of X_H is flipped from '0' to '1' then, P should point to it and hence P should contain all 0s. This is done by erasing the content of P. To do that, reset all bits of P to '0's, one bit at a time in the background. Consider the case when the bit $x_{\log n+1}$ is flipped from '1' to '0'. Before this flip, the pointer P was pointing to $x_{\log n+1}$, the rightmost bit of X_H and hence all the bits of P were '0's. So, there is no need to initialize P. After the flip, update P by inspecting the bits of X_H in the background, one bit at a time, starting from the right. Note that, either we have to *erase* the previous content of P or *construct* P by advancing the pointer one bit at a time, not both. So, there are sufficient transitions available to update P before another bit from X_H is flipped. We now describe the increment operation in detail.

4.1 The Increment Operation

The increment algorithm for our data structure is similar to the usual increment algorithm of the BRGC. First check the parity bit. If the parity is even, then flip the rightmost bit of X and the parity bit. Otherwise, find the rightmost '1' of X. Let x_s be the rightmost '1' of the code. Read all bits of X_L. If there is no '1' in X_L then, use the pointer P to find x_s, the rightmost '1' of X_H. Flip bit x_{s+1} and the parity bit. Consider the special case when bit x_n is the only '1' in the code. The pointer P is ready and points to x_n. The increment algorithm reads all bits of X_L and then follows P to find the rightmost '1' of X_H. Flip bit x_n from '1' to '0' and reset bit R to '0'. No other changes are required. After the changes take place, there is no '1' in X_H and P is still pointing to x_n with $R =$ '0'. So, we can consider P is in the last phase of the construction mode.

The update procedure for P starts when $X_L = x_{\log n} \cdots x_1 = 10 \cdots 0$ and bit $x_{\log n+1}$, the rightmost bit of X_H, is flipped to increment the value. Reset the status bit R to '0'. If bit $x_{\log n+1}$ is flipped from '0' to '1' then, set the bit E to '1' to enter the *erase* mode.

In the *erase* mode, the bit of P to be erased is determined by the BRGC denoted by the lower $\log \log n$ bits of X_L. When P enters the *erase* mode, these bits are all '0's. By the definition of the BRGC, in the next $\log n$ transitions the contents of these lower $\log \log n$ bits go through the BRGC sequence of dimension $\log \log n$ and of length $\log n$. The status bit E is set to '0' and bit R is set to '1', once all bits of P are erased.

In the *construction* mode, P points to the bit of X_H to be inspected next, if it is not ready yet. If the bit of the code is '1', the rightmost '1' of X_H is found and the status bit R is set to '1'. Otherwise, P is incremented. So, P is a counter and the BRGC of dimension $\log n$ is used for P. As we read all bits of P, the increment of P requires only a single bit change.

So, the increment operation inspects at most $2 \log n + 4$ bits and changes at most 4 bits. The pointer P can be updated in at most $n - \log n$ steps, while we have n steps available to update P.

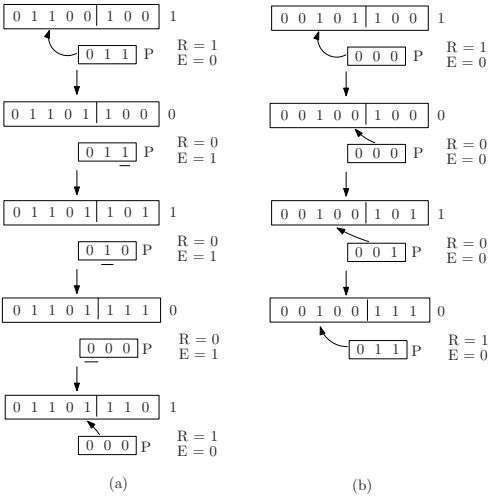


Fig. 1. Increment operations using a constant number of bit changes. (a) the pointer P in *erase* mode (b), the pointer P in *construction* mode.

4.2 Decrement Operation

The decrement algorithm for the BRGC is similar to the increment algorithm. First, inspect the parity of the code. If the parity is odd, flip the rightmost bit of the code. Otherwise, find the rightmost '1' and flip the bit to its left. The algorithm is similar to the increment algorithm except the role of the parity bit is reversed. The special case is when all bits of the code are '0's, the first code of the BRGC sequence. There is no '1' in X_H and pointer P points to bit x_n. In this case, flip bit x_n and set bit R to '1' as P is ready without entering erase or construction mode.

Consider the situation that supports both increment and decrement operation. The complication arises when the pointer P enters the *erase* mode. In the *erase* mode, the lower $\log \log n$ bits of the code denotes which bit of the pointer P to be erased. When only the increment or only the decrement is supported, these bits go through a counting sequence of length $\log n$ and can be used to point to the bit of P to be erased. Even though this counting sequence does not exist during a mix of increment and decrement operations, the increment or the decrement operation remains unaffected.

Consider the situation, when we start from a BRGC code that is obtained by the flip of bit $x_{\log n+1}$ from the previous code and the pointer P enters the *erase* mode. Due to the reflected property of BRGC, the i-th bit of P is erased before the $(i+1)$-st bit. The pointer P remains in the *erase* mode as long as the last bit of P doesn't get a chance to be erased. But in between, only bit x_i is flipped each time, for some i such that $\log \log n > i$. The pointer P needs to be ready only when a bit in X_H is to be flipped. By that time, the content of $x_{\log \log n} \cdots x_1$ goes through all possible combinations to erase P completely. So,

the increment and the decrement has the same bit probe complexity. The result is summarized in the following theorem.

Theorem 2. *An integer of dimension n can be represented by a data structure that uses $n + \log n + 3$ bits so that the increment and the decrement operations require at most $2\log n + 4$ bit inspections and at most 4 bit changes per operation.*

5 Increment and Decrement Using Fewer Bit Inspections

Next we modify our data structure to support the increment and decrement operations using fewer bit inspections. Our new data structure reduces the number of bit inspections from $2\log n + 4$ to $\log n + O(\log\log n)$. The following property of BRGC is important in designing the data structure.

> *Observation*: Let $G_i(n)$ be the subset of $G(n)$ containing the initial code of $G(n)$ and the codes of $G(n)$ that are obtained by flipping the j-th bit from the previous code, for all i and j such that $j \geq i \geq 1$. The sequence $G_i(n)$ is the BRGC sequence $G(n - i + 1)$, if the rightmost $(i - 1)$ bits from each code are discarded. An example is shown in Figure 2.

0 0 0 0	**1 1 0 0**		
0 0 0 1	1 1 0 1		
0 0 1 1	1 1 1 1		
0 0 1 0	1 1 1 0		
0 1 1 0	**1 0 1 0**	**0 0** 0 0	0 0
0 1 1 1	1 0 1 1	**0 1** 1 0	0 1
0 1 0 1	1 0 0 1	**1 1** 0 0	1 1
0 1 0 0	1 0 0 0	**1 0** 1 0	1 0
	(a)	(b)	(c)

Fig. 2. (a) The BRGC sequence $G(4)$. The codes in bold forms $G_3(4)$. (b) The subset $G_3(4)$. (c) Discarding the rightmost 2 bits from $G_3(4)$ gives us $G(2)$.

The dimension n BRGC code is divided into k blocks and denoted by $X = X_k \cdots X_1$. The block X_i contains n_i bits and denoted by $X_i = x_{l_i} \cdots x_{l_{i-1}+1}$, where $l_0 = 0$ and $l_i = \sum_{j=1}^{i} n_j$, for all i such that $k \geq i \geq 1$. Block X_1 contains a constant number of bits and the sizes of the blocks are related by $n_i = 2^{n_{i-1}}$, for all i such that $k \geq i > 1$. We have $\sum_{i=1}^{k} n_i = n$, and hence $k = \log^* n$.

A pointer P_i is associated with a block X_i along with two status bits E_i and R_i, for all i such that $k \geq i > 1$. Store the parity of X explicitly. When a pointer P_i is ready, it points to the rightmost '1' of X_i. If there is no '1' in X_i then, pointer P_i points to x_{l_i} after scanning all bits of X_i. So the size of a pointer P_i is $|P_i| \geq \log n_i = n_{i-1}$. Total number of bits used by the pointers is $\sum_{i=2}^{k} |P_i| = \sum_{i=2}^{k} n_{i-1} = n_{k-1} + n_{k-2} + \sum_{i=1}^{k-3} n_i \leq \log n + \log\log n + \sum_{i=1}^{k-3} n_i = \log n + O(\log\log n)$. The status bits use $2(k-1) = O(\log^* n)$ bits. So, the total size of the data structure is $n + \log n + O(\log\log n)$ bits.

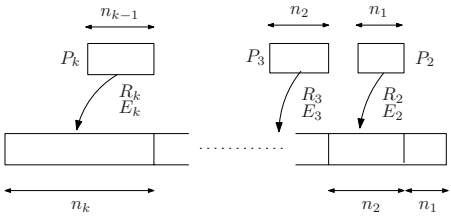


Fig. 3. Data structure for integer representation that uses fewer bit inspections per operation

5.1 Increment and Decrement

During a sequence of increment operations, consider the transitions that changes the bits only in X_i and X_{i-1}, for $k \geq i > 1$. From the observations, there are $2^{n_{i-1}}$ transitions between two bit changes in X_i, where bits in X_{i-1} are flipped. So, when a pointer P_i becomes invalid, we can use these $2^{n_{i-1}}$ transitions to reconstruct P_i in the background. Using a similar argument, as used in the previous section, we can show that a pointer P_i is always ready when it is needed during a mix of increment and decrement operations.

To increment, first read the parity bit. If the parity is even, the rightmost bit x_1 is to be flipped. Otherwise, read all the bits of X_1. If there is at least a '1' in X_1 then, we found x_s, the rightmost '1' of X. Otherwise, there is no '1' in X_1. In that case, find the rightmost block X_i that contains at least a '1'. To do that, read the status bits E_j and R_j, for $i \geq j > 1$. Note that, there is no '1' in X_j and none of the pointers P_j is in the *erase* mode. If P_j is not ready, then it points to bit x_{l_j}, the last bit of X_j. Read P_j and bit x_{l_j} to confirm that X_j contains no '1'. By the properties mentioned earlier, P_i either must be ready and points to the rightmost '1' of X_i or points to the last bit of X_i and all lower order bits of X_i are '0's. Read the pointer P_j to get x_s the rightmost '1'.

For the special case, x_s is the bit x_n. Flip bit x_n and set bit R_k to '0'. Otherwise, bit x_{s+1} needs to be flipped to complete the operation. Let $x_{s+1} \in X_t$, where $k \geq t \geq 1$. Before flipping bit x_{s+1}, process pointer P_{t+1}, if it is not ready. Read the status bits E_{t+1} and R_{t+1} (only when $t < k$) and continue the update procedure of P_{t+1}, if necessary. To erase a bit of P_{t+1}, read the rightmost $\log n_t$ bits of X_t, and erase the corresponding bit of P_{t+1}. Finally, flip the parity bit and bit x_{s+1}. If x_{s+1} is the rightmost bit of X_t then, switch the pointer P_t into *erase* mode or *construction* mode depending on the value of x_{s+1} by changing the status bits E_t and R_t appropriately. So, in the worst case, we might have to read all the pointers. In total, we have to read at most $\sum_{i=2}^{k} n_{i-1} + O(k) = \log n + O(\log \log n)$ bits. The number of bit changes required in the worst case is 6. It happens when P_{t+1} switches from *erase* mode to *ready* mode and at the same time P_t enters *erase* mode. The decrement operation is similar. The results can be summarized into the following theorem.

Theorem 3. *An integer of dimension n can be represented by a data structure that uses $n+\log n+O(\log\log n)$ bits so that increment and decrement operations require at most $\log n+O(\log\log n)$ bit inspections and 6 bit changes per operation.*

6 Supporting Addition and Subtraction

The natural extensions of increment and decrement operations are addition and subtraction. We consider the addition operation of the form $X \leftarrow (X+Y) \bmod 2^n$, where X and Y are BRGC of dimension n and m respectively such that $n > m$. First, we review the serial addition algorithm [11] of form $S \leftarrow X+Y$ for the BRGC, where $X = x_n \cdots x_1$, $Y = y_n \cdots y_1$, and $S = s_n \cdots s_1$ are three BRGC of dimension n.

Addition

$E_0 \leftarrow$ parity of X
$F_0 \leftarrow$ parity of Y
for $i := 1$ to n do
  $s_i \leftarrow (E_{i-1} \wedge F_{i-1}) \oplus x_i \oplus y_i$
  $E_i \leftarrow (E_{i-1} \wedge \neg F_{i-1}) \oplus x_i$
  $F_i \leftarrow (\neg E_{i-1} \wedge F_{i-1}) \oplus y_i$
end for

At the end of the loop, we must have $E_n = 0$ and $F_n = 0$ to get a valid addition. Otherwise, there is an overflow.

6.1 Addition with Different Size Operands

Let X and Y have dimensions n and m respectively with $n > m$. The code Y can be padded with '0's to make it a code of length n. In that case $y_i = 0$, for $n \geq i > m$. The addition can be performed in two steps — (i) $x_m \cdots x_1$ is added with $y_m \cdots y_1$ using the serial addition algorithm 6, and (ii) $x_n \cdots x_{m+1}$ is added with $y_n \cdots y_{m+1}$ quickly. We rewrite the formulae for the sum and carry bits as, $s_i = (E_{i-1} \wedge F_{i-1}) \oplus x_i$; $E_i = (E_{i-1} \wedge \neg F_{i-1}) \oplus x_i$; $F_i = \neg E_{i-1} \wedge F_{i-1}$, for all i such that $n \geq i > m$. At the end of the step (i), there can be three different cases possible based on the values of carry bits E_m and F_m.

Case 1: $F_m = 0$. We have $F_i = 0$, and $s_i = x_i$, for $n \geq i > m$. In other words, if $F_m = 0$ then, the remaining $n-m$ bits of the sum are same as those bits of X.

Case 2: $F_m = 1$ and $E_m = 1$. We have $s_{m+1} = \neg x_{m+1}, E_{m+1} = x_{m+1}$ and $F_{m+1} = 0$. As the carry bit F_{m+1} becomes '0', we know that, F_i remains '0' in the later iterations, for $n \geq i > m+1$. As a result, we have $s_i = x_i$, for $n \geq i > m+1$. In other words, if $F_m = 1$ and $E_m = 1$ then, the sum bit s_{m+1} is the opposite of bit x_{m+1}. The other bits of the sum are same as those of X.

Case 3: $F_m = 1$ and $E_m = 0$. Let x_{m+j} be the rightmost '1' of $x_n \cdots x_{m+1}$. We have, $F_{m+1} = \neg E_m \wedge F_m = 1$; $E_{m+1} = (E_m \wedge \neg F_m) \oplus x_{m+1} = x_{m+1}$;

$s_{m+1} = (E_m \wedge F_m) \oplus x_{m+1} = x_{m+1}$. Proceeding in that way we can derive, $F_{m+i} = 1, E_{m+i} = x_{m+i} = 0$ and $s_{m+i} = x_{m+i}$, for $j > i \geq 1$. Also, we have $F_{m+j} = 1, E_{m+j} = x_{m+j} = 1$ and $s_{m+j} = x_{m+j}$. This is similar to case 2 and we get the formula $s_{m+j+1} = \neg x_{m+j}$, and $s_{m+j+t} = x_{m+j+t}$, for $t > 1$. Combining all the formulae, we have, $s_{m+i} = x_{m+i}$, for $j \geq i \geq 1$; $s_{m+j+1} = \neg x_{m+j+1}$; and $s_{m+j+t} = x_{m+j+t}$, for $t > 1$. Overflow occurs when $m + j = n$. In that case, we have $s_n = \neg x_n$. In other words, when $E_m = 0$ and $F_m = 1$, copy the bits $x_n \cdots x_{m+1}$ into $s_n \cdots s_{m+1}$, find the rightmost '1' of $s_n \cdots s_{m+1}$ and flip the bit to its left. If s_n is that '1' then, flip s_n.

6.2 The Data Structure

The data structure used in our first solution is not suitable to perform addition operation of the form $X \leftarrow (X + Y) \bmod 2^n$ efficiently. The pointer P points to x_j, the rightmost '1' of X_H. If $j > m$ then, addition can be done efficiently, as P points to the rightmost '1' of $x_n \cdots x_{m+1}$. Otherwise, P can not help in finding the rightmost '1' of $x_n \cdots x_{m+1}$. In worst case, we have to inspect all bits of X_H. So, the addition requires $O(n + m)$ bits inspections and $O(m)$ bit changes. For similar reason, the data structure used in the previous section does not support efficient addition/subtraction operation.

We now present our data structure and show that the increment/decrement operations have the same complexity as the other two solutions. The n-bit BRGC number is divided into $k + 1$ blocks denoted by $X = X_k, \cdots, X_1, X_0$. Let n_i be the number of bits in the block X_i and is denoted by $X_i = x_{l_i} \cdots x_{l_{i-1}+1}$, where $l_{-1} = 0$ and $l_i = \sum_{j=0}^{i} n_j$, for $k \geq i \geq 0$, such that $n_0 = n_1 = \log n$, and $n_i = 2n_{i-1}$, for $k \geq i > 1$. Since $\sum_{i=1}^{k} n_i = n$, we have $k = \log n - \log\log n$. With each block X_i, for $k \geq i \geq 1$, there is a pointer P_i that points to the rightmost '1' of X_i, when it is ready. In addition to E_i and R_i, each block X_i is associated with one more status bit Z_i that denotes whether or not there is a '1' in X_i. Total space used by the data structure is $n + 1 + \sum_{i=1}^{k} \log n_i + 3k = n + O((\log n)^2)$ bits. The increment and the decrement operations are similar as before. It can be shown that increment/decrement operations require $3\log n - 2\log\log n + O(1) = O(\log n)$ bit inspections and at most 5 bit changes.

6.3 Addition and Subtraction

Let t be the smallest integer such that $l_t \geq m > l_{t-1}$. Addition proceeds in two steps — (i) add $x_m \cdots x_1$ with $y_m \cdots y_1$ using the serial addition algorithm 6. During this process construct the pointers P_i and status bits E_i, R_i and Z_i, for $t \geq i \geq 1$. Based on the value of the carry bits determine whether there is a carry to forward. If not, we are done. (ii) Otherwise, find the rightmost '1' of $x_n \cdots x_{m+1}$, denoted by x_s, and flip bit x_{s+1}. If x_n is that '1', then flip x_n.

To find x_s, the rightmost '1' of $x_n \cdots x_{m+1}$, read bits $x_{l_t} \cdots x_{m+1}$ first. If there is any '1', we are done. Otherwise, skip all blocks with no '1' bits by checking the zero status bits of these blocks. Let X_j be the rightmost block with some

'1' such that $k \geq j > t$. We claim that the pointer P_j is valid and points to the rightmost '1' of X_j. The pointer P_j becomes invalid when the rightmost bit of X_j is flipped. It might take at least n_j changes in X_{j-1} to complete the update of P_j. A problem might arise when there is an addition operation before P_j is updated completely. But when P_j becomes invalid, bit $x_{l_{j-1}}$ the last bit of X_{j-1}, is '1'. It requires at least $2^{n_{j-1}-1} > n_j$ transitions to get all '0's in X_{j-1}, unless step (i) affects X_{j-1}. In that case, the situation is handled by reading all bits of X_j, without increasing the asymptotic complexity of the operation.

If bit x_s is the bit x_n, then flip this bit. Otherwise, let bit x_{s+1} be in X_r for some integer $r \geq t$. Before flipping bit x_{s+1}, check the status of P_{r+1}. Perform the maintenance operations (in *erase* or *construction* mode) of P_{r+1} if necessary. Finally, flip bit x_{s+1}.

The only remaining concern is the update of a pointer in the background. There are enough steps to update an invalid pointer when only increment or decrement operations are supported. We now show that even with the addition operation, there remain enough steps to rebuild a pointer. Note that only P_{t+1} might be affected adversely after an addition operation. Read all bits of X_{t+1}, in case of an invalid pointer P_{t+1}, and rebuild it from scratch. So, the addition operation requires $O(m + \log n)$ bit probes. The subtraction operation is similar except that bit F_0 is set to the inverse of the parity of Y. Hence, we have:

Theorem 4. *An integer of dimension n can be represented by a data structure that uses $n + O((\log n)^2)$ bits so that increment and decrement operations require $O(\log n)$ bit inspections and at most 5 bit changes per operation. Addition and subtraction between two integers of dimension n and m respectively with $n > m$ requires $O(m + \log n)$ bit probes per operation.*

7 Conclusion

We have examined the problem of integer representation using a nearly minimum space with few bit inspections and changes for some basic operations. The problem is only for study in the bit probe model, as in the cell probe model we get trivial constant time solutions. We proved a new lower bound of $\Omega(\sqrt{n})$ bit probes in the worst case for the increment operation using exactly n bits to store an integer of dimension n. Our conjecture on the lower bound for the increment operation is $\Omega(n)$. By exhaustive searching we can show that the lower bound holds for a small n. Future goal is to find a lower bound of $\Omega(n)$ bit probes per operation of a counting sequence of dimension n using no extra space.

The addition of a moderate amount of extra space speeds up the operations. We present several data structures that uses little extra space for efficient increment/decrement and addition/subtraction operations. Our first solution uses $\log n + 3$ extra bits that requires at most $2 \log n + 4$ bit inspections and at most 4 bit changes for the increment/decrement operation. The second structure uses $\log n + O(\log \log n)$ extra bits that uses at most $\log n + O(\log \log n)$ bit inspections and 6 bit changes for the increment/decrement operation. Although these two structures support efficient increment/decrement operation,

they are not suitable for efficient addition/subtraction. Our third data structure requires $O((\log n)^2)$ additional bits to represent an integer of dimension n. The increment/decrement operation on this structure has the same asymptotic complexity. The addition/subtraction between two numbers of dimensions n and m with $n > m$, uses $O(m + \log n)$ bit probes in the worst case. All these data structures use exponentially less space, compared to the best known previous results, while improving or retaining the same time complexity for the operations on them.

References

1. Boothroyd, J.: Algorithm 246 Graycode. Communications of the ACM 7(12), 701 (1964)
2. Buhrman, H., Miltersen, P.B., Radhakrishnan, J., Venkatesh, S.: Are Bitvectors Optimal? SIAM Journal on Computing 31(6), 1723–1744 (2002)
3. Carlsson, S., Munro, J.I., Poblete, P.V.: An Implicit Priority Queue with Constant Insertion Time. In: Karlsson, R., Lingas, A. (eds.) SWAT 1988. LNCS, vol. 318, pp. 1–13. Springer, Heidelberg (1988)
4. Clancy, M.J., Knuth, D.E.: A Programming Problem-Solving Seminar. Tech Report, Computer Science Dept. School of Humanities and Science, Stanford University. STAN-CS-77-606 (1977)
5. Doran, M.W.: The Gray Code. CDMTCS Research Report (2007), `www.cs.auckland.ac.nz/CDMTCS//researchreports/304bob.pdf`
6. Er, M.C.: Remark on Algorithm 246 (Gray Code). ACM Transactions on Mathematical Software 11(4), 441–443 (1985)
7. Erdös, P., Rado, R.: Intersection Theorems for Systems of Sets. Journal of the London Mathematical Society 35, 85–90 (1960)
8. Frandsen, G.S., Miltersen, P.B., Skyum, S.: Dynamic Word Problems. Journal of the ACM 44, 257–271 (1997)
9. Fredman, M.L.: Observations on the Complexity of Generating Quasi-Gray Codes. SIAM Journal on Computing 7, 134–146 (1978)
10. Gray, F.: Pulse Code Communications. U.S. Patent 2632058 (1953)
11. Lucal, H.: Arithmetic Operations for Digital Computers Using a Modified Reflected Binary Code. IEEE Transactions on Computers, 449–458 (1959)
12. Miltersen, P.B.: Lower Bounds on the Size of Selection and Rank Indexes. In: Proceedings of the 16th ACM/SIAM SODA, pp. 11–12 (2005)
13. Minsky, M., Papert, S.: Perceptrons. MIT Press, Cambridge (1969)
14. Misra, J.: Remark on Algorithm 246:Graycode[Z]. ACM Transactions on Mathematical Software 1(3), 285 (1975)
15. Pătraşcu, C.E., Pătraşcu, M.: On Dynamic Bit-Probe Complexity. In: Caires, L., Italiano, G.F., Monteiro, L., Palamidessi, C., Yung, M. (eds.) ICALP 2005. LNCS, vol. 3580, pp. 969–981. Springer, Heidelberg (2005)
16. Radhakrishnan, J., Raman, V., Rao, S S.: Explicit Deterministic Constructions for Membership in the Bitprobe Model. In: Meyer auf der Heide, F. (ed.) ESA 2001. LNCS, vol. 2161, pp. 290–299. Springer, Heidelberg (2001)
17. Yao, A.C.C.: Should Tables be Sorted? Journal of The ACM 28, 615–628 (1981)

Minimum Degree Orderings*

Hiroshi Nagamochi

Department of Applied Mathematics and Physics,
Kyoto University
nag@amp.i.kyoto-u.ac.jp

Abstract. It is known that, given an edge-weighted graph, a maximum adjacency ordering (MA ordering) of vertices can find a special pair of vertices, called a *pendent pair*, and that a minimum cut in a graph can be found by repeatedly contracting a pendent pair, yielding one of the fastest and simplest minimum cut algorithms. In this paper, we provide another ordering of vertices, called a minimum degree ordering (MD ordering) as a new fundamental tool to analyze the structure of graphs. We prove that an MD ordering finds a different type of special pair of vertices, called a *flat pair*, which actually can be obtained as the last two vertices after repeatedly removing a vertex with the minimum degree. By contracting flat pairs, we can find not only a minimum cut but also all extreme subsets of a given graph. These results can be extended to the problem of finding extreme subsets in symmetric submodular set functions.

1 Introduction

Let $\Re$ and $\Re_+$ denote the sets of reals and nonnegative reals, respectively. Let V be a finite set, where we denote $n = |V|$. A singleton set $\{v\}$ is called *trivial* and may be written as v. Let $(G = (V, E), w)$ be a hypergraph with vertex set V, edge set E and weight function $w : E \mapsto \Re_+$. The *cut function* of (G, w) is defined by set function $d_{(G,w)} : 2^V \mapsto \Re^+$ such that $d_{(G,w)}(X) = \sum\{w(e) \mid e \in E,\ e \cap X \neq \emptyset \neq e - X\}$. For two specified vertices u and v, let $\lambda_{(G,w)}(u, v)$ denote the local-edge-connectivity $\min\{d_{(G,w)}(X) \mid u \in X \subseteq V - v\}$. A hypergraph $G = (V, E)$ is called a *graph* if $|e| = 2$ for all $e \in E$.

Analyzing the connectivity structure of a given graph is an important research issue, and several compact representations of connectivity structure of graphs such as Gomory-Hu trees [8], cactus representations [3] and extreme subsets [22] have been discovered. These representations have numerous applications for designing efficient graph algorithms (see [11]).

Computing a minimum cut X, i.e., a nonempty subset $X \subset V$ that minimizes $d_{(G,w)}$, is one of the basic problems in the issue, and has been studied extensively (see [2]). In particular, for undirected graphs, several algorithms that compute

* This is an extended abstract. This research was partially supported by the Scientific Grant-in-Aid from Ministry of Education, Culture, Sports, Science and Technology of Japan.

T. Tokuyama (Ed.): ISAAC 2007, LNCS 4835, pp. 17–28, 2007.

a minimum cut without relying on a maximum flow algorithm are known so far. These algorithms use a procedure of contracting a pair of vertices into a single vertex. Based on the procedure of choosing an edge with probability proportional to its edge weight and contracting the edge, Karger and Stein [7] gave an $O(n^2 \log n)$ time randomized algorithm to compute a minimum cut with high probability without using maximum flow computation. Afterwards, Karger [6] showed that a minimum cut can be computed in nearly linear time with high probability. Nagamochi and Ibaraki [14] first gave a minimum cut algorithm that chooses an edge to be contracted based on a structural feature of graphs.

A pair of vertices u and v is called a *pendent pair* if it satisfies

$$d_{(G,w)}(X) \geq \min\{d_{(G,w)}(u), d_{(G,w)}(v)\} \text{ for all } X \subseteq V \text{ with } |X \cap \{u, v\}| = 1. \tag{1}$$

i.e., $\lambda_{(G,w)}(u, v) = \min\{d_{(G,w)}(u), d_{(G,w)}(v)\}$. The existence of pendent pairs is implied by Gomory-Hu trees that represent the structure of all local-edge-connectivities [8]. An edge-weighted spanning tree $(T = (V, F), w')$ is called a *Gomory-Hu tree* of (G, w) if (i) $\lambda_{(T,w')}(x, y) = \lambda_{(G,w)}(x, y)$ holds for all $x, y \in V$ and (ii) for each edge $e = \{u, v\} \in F$, the two components $T_1 = (V_1, F_1)$ and $T_2 = (V_2, F_2)$ in $(V, F - e)$ satisfy $d_{(G,w)}(V_1) = d_{(G,w)}(V_2) = \lambda_{(G,w)}(u, v)$. For example, Figure 1(b) shows a Gomory-Hu tree for the graph $G = (V, E)$ in Fig. 1(a). By definition of Gomory-Hu trees, every leaf vertex u and its unique neighbour v in (T, w') give a pendent pair. In the graph (G, w) in Figure 1(a), $\{u_3, u_1\}$ and $\{u_6, u_2\}$ are pendent pairs.

Interestingly a pendent pair can be found by the following simple procedure. An ordering $\sigma = (v_1, v_2, \ldots, v_n)$ of the vertices in an edge-weighted graph (G, w) is called a *maximum adjacency ordering* (MA ordering, for short) if it satisfies

$$d_{(G,w)}(\{v_1, v_2, \ldots, v_{i-1}\}, v_i) \geq d_{(G,w)}(\{v_1, v_2, \ldots, v_{i-1}\}, v_j), \quad 2 \leq i \leq j \leq n, \tag{2}$$

where $d_{(G,w)}(X, Y)$ denotes $\sum\{w(e) \mid e = \{x, y\} \in E,\ x \in X,\ y \in Y\}$. It is shown that an MA ordering identifies a pendent pair of (G, w) as its last two vertices; i.e., it holds

$$d_{(G,w)}(v_n) = \lambda_{(G,w)}(v_{n-1}, v_n). \tag{3}$$

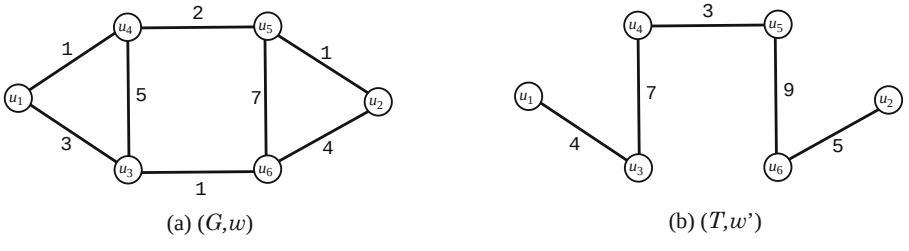


Fig. 1. (a) An edge-weighted graph $(G = (V, E), w)$ and (b) A Gomory-Hu tree $(T = (V, F), w')$ of the graph (G, w), where the numbers beside edges indicate their weights

For example, $\sigma = (u_1, u_3, u_4, u_5, u_6, u_2)$ is an MA ordering of the graph (G, w) in Figure 1(a), indicating that $\{u_6, u_2\}$ is a pendent pair.

Based on property (3), one can design an algorithm to compute a minimum cut in a graph (G, w) by repeatedly identifying a pendent pair and contract the pair into a single vertex (see [14,21]). This yields one of the fastest and simplest minimum cut algorithms, which runs in $O(nm + n^2 \log n)$ time, where $m = \sum_{e \in E} |e|$. However, no algorithm that constructs a Gomory-Hu tree of a given graph (G, w) by using MA orderings is known. Importantly Queyranne [18] extended the minimum cut algorithm for graphs to a combinatorial strongly polynomial algorithm for minimizing a *symmetric* submodular set function. Recently combinatorial strongly polynomial algorithms for minimizing general submodular set functions have been obtained by Iwata et al. [5] and Schrijver [20]. However, for minimizing symmetric submodular set functions, Queyranne's algorithm remains significantly simpler than these algorithms.

In this paper, we prove that a different structural feature of graphs can be used to design a simple and efficient connectivity algorithm. We define a new type of special pair of vertices, called a "flat pair," based on "extreme subsets" of graphs. We then introduce another ordering of vertices, called "a minimum degree ordering" (MD ordering) to identify a flat pair. A nonempty proper subset X of V is called an *extreme subset* of a graph $(G = (V, E), w)$ if

$$d_{(G,w)}(Y) > d_{(G,w)}(X) \text{ for all nonempty proper subsets } Y \text{ of } X. \tag{4}$$

We denote by $\mathcal{X}(G, w)$ the family of all extreme subsets of (G, w). Any singleton $\{v\}$, $v \in V$ is an extreme subset. Note that at least one of extreme subsets corresponds to a minimum cut, and no two extreme subsets cross each other. Figure 2(a) shows the extreme subsets of the graph (G, w) in Fig. 1(a), and Figure 2(b) shows its tree representation that indicates the inclusionwise relation among the extreme subsets in such a way that $X \in \mathcal{X}(G, w)$ is a child of $Y \in \mathcal{X}(G, w)$ if X is the largest set properly contained in Y.

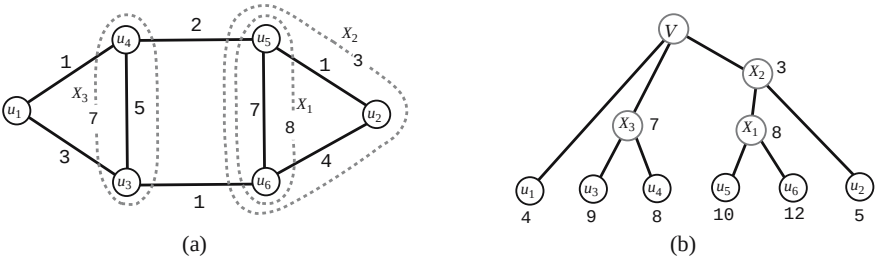


Fig. 2. (a) The extreme subsets in $\mathcal{X}(G, w)$ of the graph (G, w) in Fig. 1(a), where the numbers beside edges indicate their weights and each of the nontrivial extreme subsets $X_1, X_2, X_3 \in \mathcal{X}(G, w)$ is depicted by a dotted closed curve; (a) The tree representation for $\mathcal{X}(G, w)$

Extreme subsets are originally introduced by Watanabe and Nakamura [22] to solve the edge-connectivity augmentation problem, and extreme subsets of graphs are currently an important tool to design efficient algorithms for solving graph connectivity problems such as the source location problem [11,19], the minimum k-way cut problem [17], and the dynamic minimum cut problem [12] in addition to the connectivity augmentation problem.

In this paper, we call a pair $\{u, v\}$ in a graph (G, w), of vertices u and v a *flat pair* if it satisfies

$$d_{(G,w)}(X) \geq \min\{d_{(G,w)}(x) \mid x \in X\} \text{ for all } X \subseteq V \text{ with } |X \cap \{u, v\}| = 1. \quad (5)$$

We will prove that such a pair always exists. Observe that no nontrivial extreme subset X separates a flat pair $\{u, v\}$; i.e., $\{u, v\} \subseteq X$ or $\{u, v\} \cap X = \emptyset$ holds for any nontrivial extreme set X. A flat pair must correspond to two leaves u and v with the same parent in the tree representation of extreme subsets. In Figure 2(a), $\{u_3, u_4\}$ and $\{u_5, u_6\}$ are flat pairs.

In this paper, we call an ordering $\pi = (v_1, v_2, \ldots, v_n)$ of the vertices in a graph (G, w) a *minimum degree ordering* (MD ordering, for short) if it satisfies

$$d_{(G_{i-1},w)}(v_i) = \min\{d_{(G_{i-1},w)}(v) \mid v \in V - \{v_1, v_2, \ldots, v_{i-1}\}\} \quad 1 \leq i \leq j \leq n-1, \quad (6)$$

where $G_{i-1} = G - \{v_1, v_2, \ldots, v_{i-1}\}$ denotes the graph obtained from G by removing vertices $v_1, v_2, \ldots, v_{i-1}$ together with all edges incident to them. Thus, an MD ordering can be easily obtained just by repeatedly removing a vertex with the minimum degree in the remaining graph. For the graph in Figure 2(a), $\pi = (u_1, u_2, u_3, u_4, u_5, u_6)$ is an MD ordering.

Surprisingly the following fact holds: the last two vertices in an MD ordering gives a flat pair. We prove this, and then show that all extreme subsets of a graph (G, w) can be computed by using flat pairs in $O(mn + n^2 \log n)$ time. It is already known [11] that all extreme subsets in a graph can be computed in $O(mn + n^2 \log n)$ time by applying an MA ordering after augmenting the given graph with a new vertex and edges. However, the augmenting process is rather artificial, and no direct extension of this algorithm to the case of submodular set functions has been successful.

Orderings of (6) in a special case where G is an unweighted simple graph are known as δ-slicings [9] or smallest-last orderings [10] which are introduced to study the structure of induced subgraphs. However, the fact that MD orderings identify flat pairs was not known, and extensions to hypergraphs or set functions are not trivial.

In this paper, we define flat pairs and MD orderings in terms of set functions on V, and prove that extreme subsets of symmetric submodular set functions can be computed with the same time complexity of Queyranne's algorithm. Since one of the extreme subsets minimizes the set function, our new algorithm also solves the minimization problem for symmetric submodular set functions.

The rest of the paper is organized as follows. Section 2 introduces basic notions on submodular or posi-modular set functions, and states our main result in terms of set functions. Section 3 then proves that the last two elements in an MD

ordering in a symmetric and crossing submodular or intersecting posi-modular set function is a flat pair. Section 4 gives an algorithm that computes all extreme subsets in such a set function. Section 5 makes concluding remarks.

2 Preliminaries

Let V be a finite set, where we denote $n = |V|$. A singleton set $\{v\}$ is called *trivial* and may be written as v. The union of a set X and a singleton $\{v\}$ may be written as $X + v$. A subset $X \subseteq V$ *separates* two elements $u, v \in V$ if $|X \cap \{u, v\}| = 1$. For two subsets $X, Y \subseteq V$, we say that X and Y *intersect* each other if $X \cap Y \neq \emptyset$, $X - Y \neq \emptyset$ and $Y - X \neq \emptyset$ hold, and say that X and Y *cross* each other if, in addition, $V - (X \cup Y) \neq \emptyset$ holds. A family $\mathcal{X} \subseteq 2^V$ of subsets of V is called *laminar* if no two subsets in $\mathcal{X}$ intersect each other.

A *set function* f on V is a function $f : 2^V \to \Re$. A set function f is called *fully* (resp., *intersecting, crossing*) *submodular* if

$$f(X) + f(Y) \geq f(X \cap Y) + f(X \cup Y) \tag{7}$$

holds for every (resp., intersecting, crossing) pair of sets $X, Y \subseteq V$.A set function f is called *fully* (resp., *intersecting, crossing*) *posi-modular* if

$$f(X) + f(Y) \geq f(X - Y) + f(Y - X) \tag{8}$$

holds for every (resp., intersecting, crossing) pair of sets $X, Y \subseteq V$ [15]. Notice that the class of crossing submodular set functions is wider than that of fully submodular set functions. A set function f is called *symmetric* if

$$f(X) = f(V - X) \text{ for all } X \subseteq V. \tag{9}$$

Every symmetric and fully (resp., intersecting, crossing) submodular set function is fully (resp., intersecting, crossing) posi-modular.

Given a set function f on V the set function f' obtained by *contracting* two elements $x, y \in V$ into a new element z is defined by $V' = (V - \{x, y\}) \cup \{z\}$ and

$$f'(X) = \begin{cases} f(X) & \text{if } z \notin X \subseteq V' \\ f((X - z) \cup \{x, y\}) & \text{if } z \in X \subseteq V'. \end{cases}$$

A nonempty proper subset X of V is called an *extreme subset* of f if

$$f(Y) > f(X) \text{ for all nonempty proper subsets } Y \text{ of } X.$$

We denote by $\mathcal{X}(f)$ the family of all extreme subsets of a set function f. Any trivial set $\{v\}$, $v \in V$ is an extreme subset. By definition, any nonempty subset X contains an extreme subset X' with $f(X') \leq f(X)$. In particular, $\mathcal{X}(f)$ contains a *minimizer* of f, i.e., a subset X with $f(X) = \min_{Y \in 2^V - \{\emptyset, V\}} f(Y)$. We can observe the following property on extreme subsets of a set function (the proof is omitted for space reasons).

Lemma 1. *Let f be a set function on a finite set V, and $\mathcal{X}(f)$ be the family of extreme subsets of f.*

(i) *If f is intersecting posi-modular or symmetric and crossing submodular, then $\mathcal{X}(f)$ is laminar.*
(ii) *There is a crossing posi-modular set function f such that $\mathcal{X}(f)$ is not laminar.*
(iii) *There is an asymmetric and fully submodular set function f such that $\mathcal{X}(f)$ is not laminar.* □

For a set function f on a set V, let T_f denote the time to evaluate the function value $f(X)$ of a given subset $X \subseteq V$. In this paper, we prove the next result, which also solves the minimization of f since $\mathcal{X}(f)$ contains a minimizer of f.

Theorem 1. *Let f be a set function on V with $n = |V| \geq 2$. If f is symmetric and crossing submodular or intersecting submodular and posi-modular, then the family $\mathcal{X}(f)$ of extreme subsets of f can be found in $O(n^3 T_f)$ time.* □

An important example of symmetric and fully submodular functions is the cut functions of hypergraphs. Let $(G = (V, E), w)$ be a hypergraph with vertex set V, hyperedge set E $(\subseteq 2^V - (\{\emptyset\} \cup \{\{v\} \mid v \in V\}))$ and weight function $w : E \mapsto \Re_+$. The *cut function* of (G, w) is defined by $d_{(G,w)} : 2^V \mapsto \Re^+$ such that

$$d_{(G,w)}(X) = \sum \{w(e) \mid e \in E,\ e \cap X \neq \emptyset \neq e - X\}, \tag{10}$$

where we let $d_{(G,w)}(\emptyset) = d_{(G,w)}(V) = 0$. We see that $d_{(G,w)}$ is symmetric and fully submodular. Figure 2(a) illustrates all extreme subsets in $\mathcal{X}(d_{(G,w)})$ for the cut function $d_{(G,w)}$ of an edge-weighted graph $(G = (V, E), w)$.

3 Minimum Degree Orderings

To show Theorem 1, this section will introduce a new ordering of V for set functions f. Before showing this, we first review a related ordering, called a maximum adjacency ordering. A pair of elements $u, v \in V$ is called a *pendent pair* of f if

$$f(X) \geq \min\{f(u), f(v)\} \text{ for all subsets } X \text{ that separate } u \text{ and } v.$$

Given a set function f on V with $n = |V| \geq 2$, an ordering $\sigma = (v_1, v_2, \ldots, v_n)$ is a *maximum adjacency ordering* (MA ordering, for short) of V if it satisfies

$$f(v_i) - f(V_{i-1} + v_i) \geq f(v_j) - f(V_{i-1} + v_j), \quad 1 \leq i \leq j \leq n, \tag{11}$$

where $V_0 = \emptyset$ and $V_i = \{v_1, v_2, \ldots, v_i\}$ $(1 \leq i \leq n-1)$.

Queyranne [18] obtained the following result.

Theorem 2. [18] *For a given symmetric and crossing submodular function f on V with $n = |V| \geq 2$, let $\sigma = (v_1, v_2, \ldots, v_n)$, be an MA ordering of V. Then the last two elements v_{n-1} and v_n give a pendent pair.* □

Based on this, the following results are known.

Theorem 3. [18] *For a given symmetric and crossing submodular set function f on V with $n = |V| \geq 2$, a set $X \in 2^V - \{\emptyset, V\}$ that minimizes f can be found in $O(n^3 T_f)$ time.* □

Theorem 4. [15] *For a given intersecting submodular and posi-modular set function f on V with $n = |V| \geq 2$, a set $X \in 2^V - \{\emptyset, V\}$ that minimizes f can be found in $O(n^3 T_f)$ time.* □

In this paper, we introduce a new pair and a new ordering of V for set functions f. We call a pair of elements $u, v \in V$ a *flat pair* of f if

$$f(X) \geq \min_{x \in X} f(x) \text{ for all subsets } X \text{ that separate } u \text{ and } v. \tag{12}$$

Given a set function f on V with $n = |V| \geq 2$, we call an ordering $\pi = (v_1, v_2, \ldots, v_n)$ a *minimum degree ordering* (MD ordering, for short) of V if it satisfies

$$f(v_i) + f(V_{i-1} + v_i) \leq f(v_j) + f(V_{i-1} + v_j), \quad 1 \leq i \leq j \leq n, \tag{13}$$

where $V_0 = \emptyset$ and $V_i = \{v_1, v_2, \ldots, v_i\}$ $(1 \leq i \leq n-1)$. It is not difficult to see that, after choosing V_{i-1}, the next element v_i can be chosen from $V - V_i$ by evaluating $f(v) + f(V_{i-1} + v)$ for all $v \in V - V_{i-1}$ and that an MD ordering can be found in $O(n^2 T_f)$ time.

We here consider the time complexity for computing an MD ordering of the cut function $d_{(G,w)}$ of (10) in an edge-weighted hypergraph $(G = (V, E), w)$. For this, we define induced hypergraphs as follows. For a subset $X \subseteq V$, the hypergraph $G\langle X \rangle$ induced by X is defined to be an edge-weighted hypergraph $(X, A_X \cup B_X)$ with an edge weight function $w_X : E \to \Re_+$ such that

$$A_X = \{e \in E \mid e \subseteq X\}, \quad B_X = \{e - X \mid e \in E, e - X \neq \emptyset, |e \cap X| \geq 2\},$$

$$w_X(e) = \begin{cases} w(e) & \text{if } e \in A_X \\ w(e)/2 & \text{if } e \in B_X. \end{cases}$$

Note that if G contains only graph edges (i.e., $|e| = 2$, $e \in E$) then $B_X = \emptyset$. Then we can obtain the following (the proof is omitted for space reasons).

Lemma 2. *For an edge-weighted hypergraph $(G = (V, E), w)$, an ordering $\pi = (v_1, v_2, \ldots, v_n)$ such that*

$$d_{(G\langle V - V_{i-1} \rangle, w_{V - V_{i-1}})}(v_i) = \min\{d_{(G\langle V - V_{i-1} \rangle, w_{V - V_{i-1}})}(v) \mid v \in V - V_{i-1}\}, \quad i = 1, 2, \ldots, n-1 \tag{14}$$

is an MD ordering of the cut function d of G. An MD ordering π of d can be found in $O(m + n \log n)$ time and $O(m + n)$ space, where $n = |V|$ and $m = \sum_{e \in E} |e|$. □

As an analogous result to Theorem 2, we show that there exists a flat pair in a symmetric and crossing submodular function f and that it can be found by an MD ordering of f.

Theorem 5. *For a symmetric and crossing submodular set function f on V with $n = |V| \geq 2$, let $\pi = (v_1, v_2, \ldots, v_n)$ be an MD ordering of V. Then the last two vertices v_{n-1} and v_n give a flat pair.* □

We prove Theorem 5 after showing a lemma.

Lemma 3. *Let f be a symmetric and crossing submodular set function on V. For a subset $Z \subset V$, let g be a set function on $V-Z$ such that $g(X) = f(X) + f(Z \cup X)$, $X \subseteq V-Z$. Then g is symmetric and crossing submodular.*

Proof. Let X be an arbitrary subset of $V-Z$. We show that $g(X) = g((V-Z)-X)$. By definition of g, we have $g((V-Z)-X)) = f((V-Z)-X)) + f(Z \cup ((V-Z)-X)) = f(V-(Z \cup X)) + f(V-X)$, which is $f(Z \cup X) + f(X) = g(X)$ by symmetry of f. Hence g is symmetric. For two crossing subsets $X, Y \subseteq V-Z$, we have by the submodularity of f

$$\begin{aligned} g(X) + g(Y) &= f(X) + f(Y) + f(Z \cup X) + f(Z \cup Y) \\ &\geq f(X \cap Y) + f(X \cup Y) + f(Z \cup (X \cap Y)) + f(Z \cup X \cup Y) \\ &\geq g(X \cap Y) + g(X \cup Y). \end{aligned}$$

Therefore g is crossing submodular. □

Proof of Theorem 5. For each $i = 0, 1, \ldots, n-2$, we define set function f_i of $V-X_i$ by

$$f_i(X) = f(X) + f(V_i \cup X),\ X \subseteq V-V_i,$$

which is symmetric and crossing submodular on V–V_i by Lemma 3. By induction on $i = n-2, n-3, \ldots, 1, 0$, we prove that

$$f_i(X) \geq \min_{x \in X} f_i(x) \text{ for all } X \subseteq V-V_i \text{ that separate } v_{n-1} \text{ and } v_n. \tag{15}$$

Since $f_0(X) = 2f(X)$, it suffices to show that (15) holds for $i = 0$. We easily see that (15) holds for $i = n-2$. Now we assume that (15) holds for $i = j$. We prove that (15) holds for $i = j-1$. Let X be an arbitrary subset of $V - V_{j-1}$ that separates v_{n-1} and v_n.

Case-1. $v_j \notin X$: Let $x^* = \text{argmin}\{f_{j-1}(x) \mid x \in X\}$. For two crossing sets $V_{j-1} \cup X$ and $V_j + x^*$, we have by submodularity of f

$$f(V_{j-1} \cup X) + f(V_j + x^*) \geq f(V_j \cup X) + f(V_{j-1} + x^*).$$

From this and induction hypothesis $f_j(X) \geq f_j(x^*)$, we have

$$\begin{aligned} f_{j-1}(X) - f_{j-1}(x^*) &= f(X) + f(V_{j-1} \cup X) - f(V_{j-1} + x^*) - f(x^*) \\ &\geq f(X) + f(V_j \cup X) - f(V_j + x^*) - f(x^*) \\ &= f_j(X) - f_j(x^*) \geq 0. \end{aligned}$$

Hence $f_{j-1}(X) \geq f_{j-1}(x^*) \geq \min_{x \in X} f_{j-1}(x)$.

Case-2. $v_j \in X$: By the choice of v_j, $f_{j-1}(v_j) = \min_{x \in V - V_j} f_{j-1}(x)$. Consider subset $Y = (V - V_j) - X$, which separates v_{n-1} and v_n, and hence $f_{j-1}(Y) \geq \min_{y \in Y} f_{j-1}(y)$ holds by the argument in Case-1. Since $f_{j-1}(Y) = f_{j-1}(X)$ holds by symmetry, we have $f_{j-1}(X) = f_{j-1}(Y) \geq \min_{y \in Y} f_{j-1}(y) \geq f_{j-1}(v_j) = \min_{x \in X} f_{j-1}(x)$, as required.

Therefore, (15) holds for $i = j$. This proves that (15) holds for $i = 0$, i.e., Theorem 5. □

Corollary 1. *Let V be a finite set $n = |V| \geq 2$. Every symmetric and crossing submodular function f on V such that $f(v) = k$, $v \in V$ for some $k \in \Re$ admits a pair $\{u, v\} \subseteq V$ that is pendent and flat at the same time.*

Proof. We easily see that any MD ordering π of f is also an MA ordering if $f(v) = k$, $v \in V$. Hence the last two vertices in π is pendent and flat by Theorems 2 and 5. □

There is a symmetric and crossing submodular function f which has no pair that is pendent and flat at the same time. For example, the cut function $d_{(G,w)}$ in Fig. 2(a) has no such pairs, since $\{u_3, u_4\}$ and $\{u_5, u_6\}$ are the flat pairs of $d_{(G,w)}$, but neither of them is pendent.

Corollary 2. *Let f be a set function f on V with $n = |V| \geq 2$. If f is symmetric and crossing submodular or intersecting submodular and posi-modular, then a flat pair of f can be found in $O(n^2 T_f)$ time.*

Proof. If f is symmetric and crossing submodular, then we compute an MD ordering π of f in $O(n^2 T_f)$ time and choose the pair of the last two elements in π, which is flat by Theorem 5. Consider the case where f is intersecting submodular and posi-modular, where we assume $f(\emptyset) = f(V) = -\infty$ as it does not lose the intersecting submodularity and posi-modularity of f. In this case, we work with the following set function $g : 2^{V+s} \mapsto \Re \cup \{-\infty\}$ (where s is a new element): For each $X \subseteq V + s$, let

$$g(X) = \begin{cases} f(X) & \text{if } s \notin X \\ f(V - (X - s)) & \text{if } s \in X. \end{cases} \tag{16}$$

It is known [15] that, for an intersecting submodular and posi-modular set function f on V, the above set function g is symmetric and crossing submodular on $V + s$. Let π_g be an MD ordering of g, where the first element in π_g must be s since $g(s) = f(V) = +\infty$. Then the last two elements $u, v \in V$ in π_g give a flat pair of g. We see that $\{u, v\}$ is also flat in f, since any subset $X \subseteq V$ with $f(X) < \min_{x \in X} f(x)$ would imply $g(X) < \min_{x \in X} g(x)$, contradicting that $\{u, v\}$ is flat in g. Therefore, we can find a flat pair in $O(n^2 T_f)$ time even if f is intersecting submodular and posi-modular. □

4 Computing Extreme Subsets

This section presents an algorithm for computing all extreme subsets of a set function f by using flat pairs. For any nonempty subset $Y \subseteq V$, there is an

extreme subset $Y^* \in \mathcal{X}(f)$ with $Y^* \subseteq Y$ and $f(Y^*) \leq f(Y)$. Hence we see that $f(Y) > f(X)$ for all nonempty $Y \subset X$ if and only if $f(Z) > f(X)$ for all $Z \subset X$ with $Z \in \mathcal{X}(f)$. From this observation and the fact that no nontrivial extreme subset $X \in \mathcal{X}(f)$ separates any flat pair, we obtain the following algorithm for computing all extreme subsets of a set function f that admits flat pairs.

After initializing by $\mathcal{X} := \{\{v\} \mid v \in V\}$ $(\subseteq \mathcal{X}(f))$, we repeat a procedure of contracting a flat pair $n-2$ times. Let V^i, $i = n, n-1, \ldots, 2$, be the set of elements obtained after contracting the first $n-i$ flat pairs, where $|V^i| = i$ holds. For each element $x \in V^i$, let $V[x] \subseteq V$ denote the set of all elements that have been contracted into x. We maintain the property that

$$\mathcal{X} \text{ consists of all extreme subsets } X \in \mathcal{X}(f) \text{ with } X \subseteq V[x] \text{ and } x \in V^i. \quad (17)$$

After contracting a flat pair $u^i, v^i \in V^i$ into a single element z^i, we add $V[z^i]$ to $\mathcal{X}$ if $V[z^i] \in \mathcal{X}(f)$, so that (17) holds in the resulting set $V^{i-1} = (V^i - \{u^i, v^i\}) \cup \{z^i\}$ of elements. We can test whether $V[z^i] \in \mathcal{X}(f)$ or not by checking if $f(V[z^i]) < \min_{Y \in \mathcal{X}: Y \subset V[z^i]} f(Y)$. To facilitate this test, we also maintain $\mu(x) = \min_{Y \in \mathcal{X}: Y \subset V[x]} f(Y)$, $x \in V^i$ for each $i = n, n-1, \ldots, 2$. The entire algorithm is described as follows.

Algorithm ExtremeSubsets
Input: A set function f on a finite set V, where $n = |V| \geq 2$.
Output: A laminar family $\mathcal{X} \subseteq 2^V - \{\emptyset, V\}$ of extreme subsets of f.

```
1  X := {{v} | v ∈ V};
2  Let μ(v) := f(v) for all v ∈ V;
3  V^n := V; f^n := f;
4  for i := n to 3 do
5     Find a flat pair {u^i, v^i} of f^i;
6     V^{i-1} := (V^i - {u^i, v^i}) ∪ {z^i};
7     Let f^{i-1} be the set function on V^{i-1} obtained from f^i by contracting
      elements u^i and v^i into a single element z^i;
8     Let V[z^i] ⊆ V be the set of all elements that have been contracted
      into z^i;   /* f(V[z^i]) = f^{i-1}(z^i) */
9     if f^{i-1}(z^i) < min{μ(u^i), μ(v^i)} then
10       X := X ∪ {V[z^i]};   μ(z^i) := f^{i-1}(z^i)
11    else
12       μ(z^i) := min{μ(u^i), μ(v^i)}
13    end if
14 end for
```

From the above argument, we see that algorithm ExtremeSubsets computes the set $\mathcal{X}(f)$ of all extreme sets of f correctly as long as we can always find a flat pair in line 5. If a given set function f is symmetric and crossing submodular or intersecting submodular and posi-modular, then it is not difficult to see that each set function f^i obtained from f by contracting elements remains symmetric and crossing submodular or intersecting submodular and posi-modular. Therefore,

by Corollary 2, we can find a flat pair in $O(n^2 T_f)$ time. Then the running time of ExtremeSubsets is $O(n^3 T_f)$. This establishes Theorem 1.

By Lemma 2, we easily see that, for hypergraphs, algorithm ExtremeSubsets can be implemented to run in $O(n(m + n \log n))$ time and $O(m + n)$ space.

Corollary 3. *For an edge-weighted hypergraph $(G = (V, E), w)$, the set $\mathcal{X}(d)$ of all extreme subsets can be found in $O(n(m + n \log n))$ time and $O(m + n)$ space, where $n = |V|$ and $m = \sum_{e \in E} |e|$.* □

5 Conclusion

MA orderings were originally introduced to find a forest decomposition of multigraphs in linear time by Nagamochi and Ibaraki [13]. They realized that the an MA ordering identifies a pendent pair, and based on this, they proposed an $O(nm + n^2 \log n)$ time algorithm for computing a minimum cut in an edge-weighted graph without relying on a maximum flow algorithm. The algorithm is then extended to an $O(n^3 T_f)$ time algorithm for minimizing a symmetric and crossing submodular set function by Queyranne [18] and for an intersecting posi-modular set function by Nagamochi and Ibaraki [15]. For graphs, MA orderings can be used to sparsify multigraphs [13] and to find a maximum flow between a pendent pair in an edge-weighed graphs [1]. However, for symmetric submodular or posi-modular set functions, Queyranne's and Nagamochi and Ibaraki's algorithms based on pendent pairs can find a minimizer only.

Our new algorithm beased on flat pairs can find not only a minimizer but also all extreme subsets. Interestingly, the algorithm works for the class of intersecting posi-modular or symmetric and crossing submodular set functions, which is shown by Lemma 1 to be a maximal class of set functions whose extreme subsets always form a laminar family.

References

1. Arikati, S.R., Mehlhorn, K.: A correctness certificate for the Stoer-Wagner min-cut algorithm. Information Processing Letters 70, 251–254 (1999)
2. Chekuri, C.S., Goldberg, A.V., Karger, D.R., Levine, M.S., Stein, C.: Experimental study of minimum cut algorithms. In: Proc. 8th Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 324–333 (1997)
3. Dinits, E.A., Karzanov, A.V., Lomonosov, M.V.: On the structure of a family of minimal weighted cuts in a graph. In: Fridman, A.A. (ed.) Studies in Discrete Optimization, Nauka, Moscow, pp. 290–306 (1976) (in Russian)
4. Fredman, M.L., Tarjan, R.E.: Fibonacci heaps and their uses in improved network optimization algorithms. J. ACM 34, 596–615 (1987)
5. Iwata, S., Fleischer, L., Fujishige, S.: A combinatorial strongly polynomial algorithm for minimizing submodular functions. J. ACM 48, 761–777 (2001)
6. Karger, D.R.: Minimum cuts in near-linear time. J. ACM 47, 46–76 (2000)
7. Karger, D.R., Stein, C.: A new approach to the minimum cut problems. J. ACM 43, 601–640 (1996)

8. Gomory, R.E., Hu, T.C.: Multi-terminal network flows. SIAM J. Appl. Math. 9, 551–570 (1961)
9. Matula, D.W.: k-Components, clusters, and slicings in graphs. SIAM J. Applied Mathematics 22, 459–480 (1972)
10. Matula, D.W., Beck, L.L.: Smallest-last ordering and clustering and graph coloring algorithms. J. ACM 30, 417–427 (1983)
11. Nagamochi, H.: Graph algorithms for network connectivity problems. J. Operations Research Society of Japan 47, 199–223 (2004)
12. Nagamochi, H.: Computing a minimum cut in a graph with dynamic edges incident to a designated vertex. Inst. Electron. Inform. Comm. Eng. Trans. Fundamentals E90-D, 428–431 (2007)
13. Nagamochi, H., Ibaraki, T.: A linear-time algorithm for finding a sparse k-connected spanning subgraph of a k-connected graph. Algorithmica 7, 583–596 (1992)
14. Nagamochi, H., Ibaraki, T.: Computing edge-connectivity of multigraphs and capacitated graphs. SIAM J. Discrete Mathematics 5, 54–66 (1992)
15. Nagamochi, H., Ibaraki, T.: A note on minimizing submodular functions. Information Processing Letters 67, 239–244 (1998)
16. Nagamochi, H., Ibaraki, T.: Augmenting edge-connectivity over the entire range in $\tilde{O}(nm)$ time. J. Algorithms 30, 253–301 (1999)
17. Nagamochi, H., Kamidoi, Y.: Minimum cost subpartitions in graphs. Information Processing Letters 102, 79–84 (2007)
18. Queyranne, M.: Minimizing symmetric submodular functions. Mathematical Programming 82, 3–12 (1998)
19. Sakashita, M., Makino, K., Fujishige, S.: Minimum cost source location problems with flow requirements. In: Correa, J.R., Hevia, A., Kiwi, M. (eds.) LATIN 2006. LNCS, vol. 3887, pp. 769–780. Springer, Heidelberg (2006)
20. Schrijver, A.: A combinatorial algorithm minimizing submodular function in strongly polynomial time. J. Combinatorial Theory (B) 80, 346–355 (2000)
21. Stoer, M., Wagner, F.: A simple min cut algorithm. J. ACM 44, 585–591 (1997)
22. Watanabe, T., Nakamura, A.: Edge-connectivity augmentation problems. J. Comp. System Sci. 35, 96–144 (1987)

Greedy Approximation for Source Location Problem with Vertex-Connectivity Requirements in Undirected Graphs

Toshimasa Ishii

Department of Information and Management Science,
Otaru University of Commerce,
Otaru-city Hokkaido 047-8501 Japan
ishii@res.otaru-uc.ac.jp

Abstract. Let $G = (V, E)$ be a simple undirected graph with a set V of vertices and a set E of edges. Each vertex $v \in V$ has a demand $d(v) \in Z_+$, and a cost $c(v) \in R_+$, where Z_+ and R_+ denote the set of nonnegative integers and the set of nonnegative reals, respectively. The source location problem with vertex-connectivity requirements in a given graph G asks to find a set S of vertices minimizing $\sum_{v \in S} c(v)$ such that there are at least $d(v)$ pairwise vertex-disjoint paths from S to v for each vertex $v \in V - S$. It is known that the problem is not approximable within a ratio of $O(\ln \sum_{v \in V} d(v))$, unless NP has an $O(N^{\log \log N})$-time deterministic algorithm. Also, it is known that even if every vertex has a uniform cost and $d^* = 4$ holds, then the problem is NP-hard, where $d^* = \max\{d(v) \mid v \in V\}$.

In this paper, we consider the problem in the case where every vertex has a uniform cost. We propose a simple greedy algorithm for delivering a $\max\{d^* + 1, 2d^* - 6\}$-approximate solution to the problem in $O(\min\{d^*, \sqrt{|V|}\}d^*|V|^2)$ time. Especially, in the case of $d^* \leq 4$, we give a tight analysis to show that it achieves an approximation ratio of 3. We also show the APX-hardness of the problem even restricted to $d^* \leq 4$.

1 Introduction

Problems of selecting the best location of facilities in a given network to satisfy a certain property are called *location problems* [11]. Recently, the location problems with requirements measured by a network-connectivity have been studied extensively [2, 3, 5, 7, 6, 9, 10, 13, 14, 15, 16].

Connectivity and/or flow-amount are very important factors in applications to control and design of multimedia networks. In a multimedia network, a set S of some specified network nodes, such as the so-called mirror servers, may have functions of offering the same services for users. A user at a node v can use the service by communicating with at least one node $s \in S$ through a path between s and v. The flow-amount (which is the capacity of paths between S and v) affects the maximum data amount that can be transmitted from S to a user at a node v. Also, the edge-connectivity or the vertex-connectivity between S and

T. Tokuyama (Ed.): ISAAC 2007, LNCS 4835, pp. 29–40, 2007.

v measures the robustness of the service against network failures. The concept of such connectivity and/or flow-amount between a node and a set of specified nodes was given by H. Ito [8], considering design of a reliable telephone network with plural switching apparatuses.

Given a graph, the problem of finding the best location of such a set S of vertices, called *sources*, under connectivity and/or flow-amount requirements from each vertex to S is called the *source location problem*, which is formulated as follows:

Problem 1. (Source location problems)
Input: A graph $G = (V, E)$ with a set V of vertices and a set E of edges capacitated by nonnegative reals, a cost function $c : V \to R_+$ (where R_+ denotes the set of nonnegative reals), and a demand function $d : V \to R_+$.
Output: A vertex set $S \subseteq V$ such that $\psi(S, v) \geq d(v)$ holds for every vertex $v \in V - S$ and $\sum_{v \in S} c(v)$ is the minimum, where $\psi(S, v)$ is a measurement based on the edge-connectivity, the vertex-connectivity or the flow-amount between S and a vertex v in a graph G. □

For such measurements $\psi(S, v)$, one may consider the minimum capacity $\lambda(S, v)$ of an edge cut $C \subseteq E$ that separates v from S, the minimum size $\kappa(S, v)$ of a vertex cut $C \subseteq V - S - v$ that separates S and v, or the maximum number $\hat{\kappa}(S, v)$ of paths between S and v such that no pair of paths has a common vertex in $V - v$.

Here let us review the developments in the source location problems in undirected graphs. The problem with $\psi = \lambda$ was first considered by Tamura et al. [15]. They showed that the problem with uniform costs and uniform demands can be solved in polynomial time. Also, Tamura et al. [16] showed that the case of uniform costs and general costs is polynomially solvable, while the fastest known algorithm for it achieves complexity $O(mM(n, m))$ due to Arata et al. [2], where $n = |V|$, $m = |\{\{u, v\} \mid u, v \in V\}|$, and $M(n, m)$ denotes the time for max-flow computation in the graph with n vertices and m edges. In general, Sakashita et al. [14] showed that the problem is strongly NP-hard.

For $\psi = \kappa$, Ito et al. [9] investigated the problem with uniform costs and uniform demands $d(v) = k$, presented a polynomial time algorithm in the case of $k \leq 2$, and showed the NP-hardness of the problem in the case of $k \geq 3$. They also showed that in the case of $k \leq 2$, even if a measurement $\lambda(S, v) \geq \ell$ is added, then the problem is still polynomially solvable.

For $\psi = \hat{\kappa}$, Nagamochi et al. [13] showed that the problem with uniform demands $d(v) = k$ can be solved in $O(\min\{k, \sqrt{n}\}kn^2)$ time. In [7], Ishii et al. considered the problem with uniform costs and general demands, and showed that it can be solved in linear time in the case of $d^* \leq 3$, while it is NP-hard even restricted to $d^* = 4$, where $d^* = \max\{d(v) | v \in V\}$. They also showed that if $d^* \leq 3$, then even in the case of general costs, it is also polynomially solvable [6].

Also for directed graphs, many variants of problems have been investigated (see [3, 5, 10] for $\psi = \lambda$ and [13] for $\psi = \hat{\kappa}$).

Recently, Sakashita et al.[14] showed that no problems with the above three types of connectivity requirements in undirected/directed graphs are approximable within a ratio of $O(\ln \sum_{v \in V} d(v))$, unless NP has an $O(N^{\log \log N})$-time deterministic algorithm. They also gave $(1 + \ln \sum_{v \in V} d(v))$-approximation algorithms for all such problems if the capacity and demand functions are integral.

In this paper, we focus on the problem with $\psi = \hat{\kappa}$ in undirected graphs. As shown in [14], in general, it is unlikely that it is approximable within a ratio of $O(\ln \sum_{v \in V} d(v))$. Moreover, it was shown in [7] that even if the cost function is uniform and d^* is bounded from above by a constant, the problem is NP-hard. In this paper, we show that if the cost function is uniform, then a simple greedy algorithm delivers a $\max\{d^* + 1, 2d^* - 6\}$-approximate solution in $O(\min\{d^*, \sqrt{n}\}d^* n^2)$ time; the approximation ratio is constant if d^* is bounded from above by a constant. Especially, in the case of $d^* \leq 4$, we give a tight analysis to show that it achieves an approximation ratio of 3. We also show that the problem is APX-hard even restricted to uniform costs and $d^* \leq 4$.

We here summarize our method. First, we start with the source set $S = V$. Then, we pick up vertices v, one by one, in nondecreasing order of their demands; only when $S - \{v\}$ remains to be feasible, then update $S := S - \{v\}$. It was shown in [2, 16] that for the problem with $\psi = \lambda$ and uniform costs in undirected graphs, this algorithm delivers an optimal solution. In our problem, this method may not achieve an optimal, but an approximation ratio of $\max\{d^* + 1, 2d^* - 6\}$.

The rest of the paper is organized as follows. Some definitions and preliminaries are described in Section 2. In Section 3, we describe a greedy algorithm, named GREEDY_LVSLP, for the problem, prove that it delivers a $\max\{d^* + 1, 2d^* - 6\}$-approximate solution, and discuss the time complexity of the algorithm. In Section 4, we show that in the case of $d^* \leq 4$, the solution obtained by GREEDY_LVSLP is 3-approximate and the analysis is tight for the algorithm. Also in Section 4, we show the APX-hardness of the problem restricted to $d^* \leq 4$. Finally, we give some concluding remarks in Section 5.

2 Preliminaries

Let $G = (V, E)$ be a simple undirected graph with a set V of *vertices* and a set E of *edges*, where we denote $|V|$ by n and $|E|$ by m. A singleton set $\{x\}$ may be simply written as x, and "$\subset$" implies proper inclusion while "$\subseteq$" means "$\subset$" or "$=$". A vertex set and an edge set of graph G is denoted by $V(G)$ and $E(G)$, respectively. For a vertex subset $V' \subseteq V$, $G[V']$ means the subgraph induced by V'. For a vertex set $X \subseteq V$, $N_G(X)$ is defined as the set of all vertices in $V - X$ which are adjacent to some of vertices in X. Moreover, let $N_G(\emptyset) = 0$. For a vertex set $Y \subseteq V$ and a family $\mathcal{X}$ of vertex sets, Y *covers* $\mathcal{X}$ if each $X \in \mathcal{X}$ satisfies $X \cap Y \neq \emptyset$. For a family $\mathcal{X}$ of vertex sets in V, the *frequency* of a vertex v (wrt. $\mathcal{X}$) is defined as the number of sets of $\mathcal{X}$ which includes v, and let $f(V, \mathcal{X})$ denote the maximum frequency wrt. $\mathcal{X}$ of a vertex in V.

For a vertex $v \in V$ and a vertex set $X \subseteq V - \{v\}$ in G, we denote by $\hat{\kappa}_G(X, v)$ the maximum number of paths from v to X such that no pair of paths has a

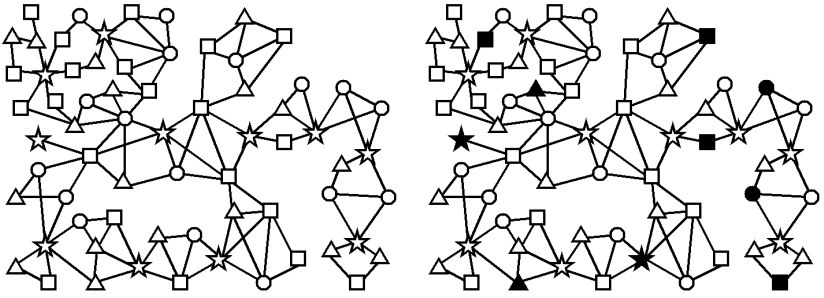


Fig. 1. Illustration of an instance of 3LVSLP. (a) An initial graph $G = (V, E)$, where each vertex $v \in V$ with $d(v) = 0, 1, 2, 3$ is drawn as a square, a triangle, a circle, and a star, respectively. (b) A set S of black vertices is a source set; there are at least $d(v)$ paths between S and each vertex $v \in V - S$ such that no pair of paths has a common vertex in $V - v$.

common vertex in $V-v$. For a vertex $v \in V$ and a vertex set $X \subseteq V$ with $v \in X$, let $\hat{\kappa}_G(X, v) = \infty$. By Menger's theorem, the following lemma holds.

Lemma 1. *For a vertex $v \in V$ and a vertex set $X \subseteq V - \{v\}$, $\hat{\kappa}_G(X, v) \geq k$ holds if and only if $|N_G(W)| \geq k$ holds for every vertex set $W \subseteq V - X$ with $v \in W$.* □

In this paper, each vertex $v \in V$ in $G = (V, E)$ has a *demand* $d(v)$ of nonnegative integer. Let $d^* = \max\{d(v) \mid v \in V\}$. A vertex set $S \subseteq V$ is called a *source set* if it satisfies

$$\hat{\kappa}_G(S, v) \geq d(v) \text{ for all vertices } v \in V - S, \tag{1}$$

and we call each vertex $v \in S$ a *source*. In this paper, we consider the following source location problem with local vertex-connectivity requirements in an undirected graph (shortly, LVSLP or d^*LVSLP). Fig. 1 gives an instance of LVSLP with $d^* = 3$.

Problem 2. (LVSLP)
Input: An undirected graph $G = (V, E)$ and a demand function $d : V \to Z^+$ (where Z^+ denotes the set of nonnegative integers).
Output: A source set $S \subseteq V$ with the minimum cardinality.

The main results of this paper are described as follows.

Theorem 1. *Given an undirected graph $G = (V, E)$ and a demand function $d : V \to Z^+$, LVSLP is $\max\{d^* + 1, 2d^* - 6\}$-approximable in $O(\min\{d^*, \sqrt{n}\}d^*n^2)$ time.* □

Theorem 2. *Let an undirected graph $G = (V, E)$ and a demand function $d : V \to \{0, 1, 2, 3, 4\}$ be given. Then a 3-approximate solution to 4LVSLP can be found in $O(n^2)$ time, while 4LVSLP is APX-hard.* □

In the rest of this section, we introduce several properties for LVSLP, which will be used in the subsequent sections. For a vertex set $X \subseteq V$, $d(X)$ denotes the maximum demand among all vertices in X, i.e., $d(X) = \max_{v \in X} d(v)$. A vertex subset $W \subseteq V$ with $d(W) > |N_G(W)|$ is called a *deficient set*. We have the following property by Lemma 1.

Lemma 2. *A vertex set $S \subseteq V$ satisfies $W \cap S \neq \emptyset$ for every deficient set W if and only if S is a source set.* □

A deficient set W is *minimal* if no proper subset of W is deficient. For a vertex $v \in V$, we say that a deficient set $W \subseteq V$ with $v \in W$ is a *minimal deficient set wrt. v*, if W is minimal deficient and $d(v) > |N_G(W)|$. A minimal deficient set has the following properties.

Lemma 3. [7] *Every minimal deficient set W wrt. $v \in W$ induces a connected graph.* □

Lemma 4. *Let W be a minimal deficient set wrt. $v \in W$. If there is a set X with $v \notin X$, $|N_G(X) \cap W| = 1$, and $X \cap N_G(W) \neq \emptyset$, then $N_G(X) \cap W = \{v\}$.*

Proof. Assume by contradiction that $v \in W - X - N_G(X)$. Now we have $N_G(W - X - N_G(X)) \subseteq (N_G(W) - X) \cup (W \cap N_G(X))$. Hence, it follows from $|N_G(W) \cap X| \geq 1$ and $|W \cap N_G(X)| = 1$ that $|N_G(W - X - N_G(X))| \leq |N_G(W) - X| + |W \cap N_G(X)| \leq |N_G(W)| - |N_G(W) \cap X| + 1 \leq |N_G(W)| < d(v)$ and $W - X - N_G(X)$ is also a deficient set, contradicting the minimality of W. □

For two vertex sets X and Y, we say that X and Y *intersect* each other, if none of $X \cap Y$, $X - Y$, and $Y - X$ is empty. For two vertex sets X and Y, the following holds.

$$|N_G(X)| + |N_G(Y)| \geq |N_G(X \cap Y)| + |N_G(X \cup Y)|. \quad (2)$$

$$|N_G(X)| + |N_G(Y)| \geq |N_G(X - Y - N_G(Y))| + |N_G(Y - X - N_G(X))|. \quad (3)$$

Lemma 5. *Let W_i, $i = 1, 2$ be a minimal deficient set wrt. $w_i \in W_i$. If W_1 and W_2 intersect each other, $w_1 \in W_1 - W_2$, and $w_2 \in W_2 - W_1$, then $w_1 \in N_G(W_2)$ or $w_2 \in N_G(W_1)$ hold.*

Proof. Assume by contradiction that $\{w_1, w_2\} \cap (N_G(W_1) \cup N_G(W_2)) = \emptyset$. By $w_1 \in W_1 - W_2 - N_G(W_2)$ and $w_2 \in W_2 - W_1 - N_G(W_1)$, we have $W_1 - W_2 - N_G(W_2) \neq \emptyset \neq W_2 - W_1 - N_G(W_2)$. Now we have $|N_G(W_1)| < d(w_1)$ and $|N_G(W_2)| < d(w_2)$. It follows from (3) that we have $d(w_1) > |N_G(W_1 - W_2 - N_G(W_2))|$ or $d(w_2) > |N_G(W_2 - W_1 - N_G(W_1))|$ (say, $d(w_1) > |N_G(W_1 - W_2 - N_G(W_2))|$). Then $W_1 - W_2 - N_G(W_2)$ is also deficient, which contradicts the minimality of W_1. Hence, it follows that $\{w_1, w_2\} \cap (N_G(W_1) \cup N_G(W_2)) \neq \emptyset$. □

3 Greedy Algorithm

For a given graph $G = (V, E)$ and a demand function $d : V \to Z^+$, let $opt(G, d)$ denote the optimal value to LVSLP. In this section, we give a simple greedy algorithm, named GREEDY_LVSLP, for finding a $\max\{d^* + 1, 2d^* - 6\}$-approximate

solution S to LVSLP in $O(\min\{d^*, \sqrt{n}\}d^*n^2)$ time. In the sequel, assume that a given graph G is connected, since if G is disconnected, then we can consider the problem separately for each connected component.

The algorithm GREEDY_LVSLP is a greedy method to find a minimal feasible solution S_0 and a family $\mathcal{W}_0$ of minimal deficient sets wrt. some $s \in S_0$. We start with the source set $S_0 = V$ and the family $\mathcal{W}_0 := \emptyset$ of minimal deficient sets, and pick up vertices $v \in V$, one by one, in nondecreasing order of their demands. If $S_0 - \{v\}$ remains to be a source set, update $S_0 := S_0 - \{v\}$, and otherwise we have a minimal deficient set W wrt. v with $W \cap S_0 = \{v\}$ and update $\mathcal{W}_0 := \mathcal{W}_0 \cup \{W\}$.

A more precise description of the algorithm is given as follows.

Algorithm GREEDY_LVSLP
Input: An undirected connected graph $G = (V, E)$ and a demand function $d : V \to Z^+$.
Output: A source set S such that $|S| \leq \max\{d^* + 1, 2d^* - 6\}opt(G, d)$.
(Step 1) Number vertices of V such as $d(v_1) \leq \cdots \leq d(v_n)$.
(Step 2) Initialize $j := 1$, $S_0 := V$, and $\mathcal{W}_0 := \emptyset$.
(Step 3) If $S_0 - \{v_j\}$ satisfies (1) then let $S_0 := S_0 - \{v_j\}$. Otherwise select a minimal deficient set $W' \subseteq V - (S_0 - \{v_j\})$ wrt. v_j, and let $\mathcal{W}_0 := \mathcal{W}_0 \cup \{W'\}$.
(Step 4) If $j < n$, then $j := j + 1$ and go to Step 3. Otherwise output S_0 as a solution. □

Note that in the case where $S_0 - \{v_j\}$ does not satisfy (1) in Step 3, there exists a minimal deficient set $W' \subseteq V - (S_0 - \{v_j\})$ wrt. v_j. Before deleting v_j from S_0, S_0 is feasible and hence by Lemma 2, every deficient set contains a source in S_0. On the other hand, $S_0 - \{v_j\}$ is infeasible. Again by Lemma 2, there is a deficient set W' with $W' \cap S_0 = \{v_j\}$ such that $W' - \{v_j\}$ is not deficient. Moreover, since all vertices in $W' - \{v_j\}$ have been already deleted, we can observe that $d(v_j) = \max\{d(v) \mid v \in W'\} = d(W')$ holds by the sorting in Step 1, and that $d(v_j) > |N_G(W')|$. It follows that there is a minimal deficient set W' wrt. v_j satisfying $W' \subseteq V - (S_0 - \{v_j\})$.

Let $S_0 = \{s_1, s_2, \ldots, s_p\}$ and $\mathcal{W}_0 = \{W_1, W_2, \ldots, W_p\}$ be a source set and a family of the deficient sets such that W_i corresponds to s_i, obtained by the algorithm, respectively. From the above observation, S_0 and $\mathcal{W}_0$ has the following properties.

Lemma 6. *Let $s \in S_0$ and $W \in \mathcal{W}_0$ be a source and the corresponding deficient set satisfying $s \in W$.*
(i) $W \cap S_0 = \{s\}$.
(ii) *W is minimal wrt. s.*
(iii) $d(W) = d(s)$.

Proof. As mentioned above, when s is not deleted from the current source set S_0' and W is chosen as a member of $\mathcal{W}_0$ in Step 3, we have $W \cap S_0' = \{s\}$ and $d(s) = \max\{d(v) \mid v \in W\} = d(W)$, and W is minimal wrt. s. Moreover, by $S_0 \subseteq S_0'$, we have $W \cap S_0 = \{s\}$. □

We can observe that S_0 is $\max\{d^* + 1, 2d^* - 6\}$-approximate.

Lemma 7. $|S_0| \leq \max\{d^* + 1, 2d^* - 6\} opt(G, d)$.

Proof. If $d^* = 1$, then $|S_0| = 1$ clearly holds; S_0 is optimal. Consider the case where $d^* \geq 2$. Let S be an arbitrary source set. From the definition of $f(V, \mathcal{W}_0)$, we can observe that S can cover at most $|S| f(V, \mathcal{W}_0)$ sets in $\mathcal{W}_0$. On the other hand, Lemma 2 indicates that we have $S \cap W \neq \emptyset$ for every $W \in \mathcal{W}_0$. Therefore, $|S| f(V, \mathcal{W}_0) \geq |\mathcal{W}_0|$ must hold. It follows that we have $opt(G, d) \geq |\mathcal{W}_0| / f(V, \mathcal{W}_0) = |S_0| / f(V, \mathcal{W}_0)$. We will prove this lemma by showing that $f(V, \mathcal{W}_0) \leq \max\{d^* + 1, 2d^* - 5\}$ and that when $f(V, \mathcal{W}_0) = 2d^* - 5$, $opt(G, d) \leq \max\{d^* + 1, 2d^* - 6\}$.

Assume that there is a family $\mathcal{W}' \subseteq \mathcal{W}_0$ of deficient sets with $|\mathcal{W}'| = \ell$, $\ell \geq d^* + 1$, and $\bigcap_{W \in \mathcal{W}'} W \neq \emptyset$. We first claim that for each $W \in \mathcal{W}'$, the number of sets $W_i \in \mathcal{W}'$ with $s_i \in N_G(W)$ is at most $d^* - 3$. From $|N_G(W)| \leq d^* - 1$, $\ell \geq d^* + 1$, and Lemma 6(i), there exists a set $W_j \in \mathcal{W}'$ with $s_j \notin N_G(W)$. Again by Lemma 6(i), if this claim would not hold, then such W_j would satisfy $|W_j \cap N_G(W)| \leq 1$. Then $|W_j \cap N_G(W)| = 0$ would imply that $W_j - W$ and

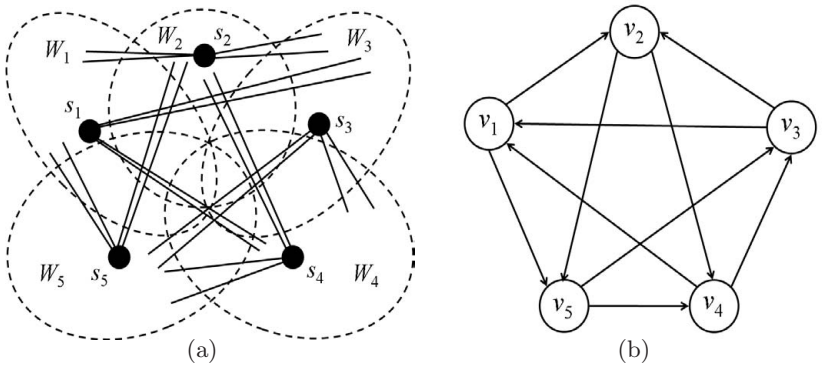


Fig. 2. Illustration of the directed graph $H = (V_1, E_1)$. (a) shows $S_0' = \{s_1, s_2, s_3, s_4, s_5\} \subseteq S_0$ and $\mathcal{W}_0' = \{W_1, W_2, W_3, W_4, W_5\} \subseteq \mathcal{W}_0$ such that $W_1 \cap W_2 \cap W_3 \cap W_4 \cap W_5 \neq \emptyset$, $S_0' \cap N_G(W_1) = \{s_2, s_5\}$, $S_0' \cap N_G(W_2) = \{s_4, s_5\}$, $S_0' \cap N_G(W_3) = \{s_1, s_2\}$, $S_0' \cap N_G(W_4) = \{s_1, s_3\}$, and $S_0' \cap N_G(W_5) = \{s_3, s_4\}$. (b) shows that the subgraph of the directed graph H corresponding to S_0' and $\mathcal{W}_0'$ (note that for each pair $W_i, W_j \in \mathcal{W}_0'$, $s_i \in N_G(W_j)$ or $s_j \in N_G(W_i)$ by Lemmas 5 and 6).

$W_j \cap W$ are disconnected, which contradicts Lemma 3, and $|W_j \cap N_G(W)| = 1$ would indicate that the vertex v with $W_j \cap N_G(W) = \{v\}$ satisfies $v = s_j$ by Lemmas 4 and 6(ii) (note that $W \cap N_G(W_j) \neq \emptyset$ holds by Lemma 3).

Consider the directed graph $H = (V_1, E_1)$ such that each vertex $v_i \in V_1$ corresponds to a set in $W_i \in \mathcal{W}'$, and that a directed edge (v_i, v_j) belongs to E_1 if and only if $s_j \in N_G(W_i)$ (see Fig. 2). From the above claim, the outdegree of each vertex in V_1 is at most $d^* - 3$. On the other hand, $|E_1| \geq \ell(\ell - 1)/2$ holds, since Lemmas 5 and 6 imply that for every two sets $W_i, W_j \in \mathcal{W}'$, we have $s_i \in N_G(W_j)$

or $s_j \in N_G(W_i)$. It follows that $(d^* - 3)\ell \geq |E_1| \geq \ell(\ell - 1)/2$; $\ell \leq 2d^* - 5$. Moreover, when $\ell = 2d^* - 5$, we can observe that we have $|N_G(W_i)| = d^* - 1$ and $N_G(W_i) \subseteq \bigcup_{W \in \mathcal{W}'} W$ for each $W_i \in \mathcal{W}'$; $N_G(\bigcup_{W \in \mathcal{W}'} W) = \emptyset$, $V = \bigcup_{W \in \mathcal{W}'} W$, $\mathcal{W}_0 = \mathcal{W}'$, and each $s_i \in S_0$ satisfies $d(s_i) = d^*$ (note that G is connected and that by $V = \bigcup_{W \in \mathcal{W}'} W$ and Lemma 6(i), any set in $\mathcal{W}_0 - \mathcal{W}'$ cannot exist). Observe that in this case, $opt(G, d) \geq 2$ since if $\{v\}$ would be an optimal solution for some $v \in V$, then $V - \{v\}$ would be a deficient set wrt. some $s \in S_0 - \{v\}$ and hence $\{v\}$ would be infeasible (note that $|N_G(V - \{v\})| = 1 < d^*$ and $|S_0| \geq d^* + 1 > 1$). It follows that if $\ell = 2d^* - 5$, $|S_0| = \ell \leq (2d^* - 5)opt(G, d)/2$. □

Finally, we show that the algorithm GREEDY_LVSLP can be implemented to run in $O(\min\{d^*, \sqrt{n}\}d^*n^2)$ time. Step 3 can be done in $O(\min\{d^*, \sqrt{n}\}m)$ time by using the network computation [4]. Since Step 3 is executed at most n times, it follows that the total complexity is $O(\min\{d^*, \sqrt{n}\}mn)$. Moreover, we can reduce this complexity to $O(m + \min\{d^*, \sqrt{n}\}d^*n^2)$ by computing a sparse subgraph G' of G with $O(d^*n)$ edges that preserves the local vertex-connectivity up to d^*. Such a sparse spanning subgraph exists and it can be computed in $O(m)$ time [12].

Summarizing the arguments given so far, Theorem 1 is now established.

4 The Case of $d^* \leq 4$

In this section, we consider the case restricted to $d^* \leq 4$. Let S_0 and $\mathcal{W}_0$ be the set of vertices and the family of deficient sets obtained by algorithm GREEDY_LVSLP, respectively, as defined in the previous section. We here show that S_0 is 3-approximate and this analysis is tight for the algorithm. We also show that 4LVSLP is APX-hard.

Assume that $d^* = 4$, since the case of $d^* \leq 3$ is polynomially solvable, as shown in [7]. Also assume that $|S_0| \geq 4$, since $|S_0| \leq 3$ implies that S_0 is 3-approximate. If the frequency of each vertex wrt. $\mathcal{W}_0$ is at most three, then S_0 is 3-approximate as observed in the proof of Lemma 7. However, there exists an instance which has a vertex with frequency four. We first start with characterizing such cases through the following preparatory lemmas.

Lemma 8. *Let W_i and W_j denote deficient sets in $\mathcal{W}_0$ with $W_i \cap W_j \neq \emptyset$.*
(i) $|N_G(W_i \cup W_j)| \geq 1$.
(ii) $W_i \cap N_G(W_j) \neq \emptyset \neq W_j \cap N_G(W_i)$ holds; $|N_G(W_i \cap W_j)| \geq 2$.
(iii) If $|N_G(W_i \cap W_j)| = 2$, then no set $W \in \mathcal{W}_0 - \{W_i, W_j\}$ satisfies $W \cap W_i \cap W_j \neq \emptyset$.
(iv) If $|N_G(W_i \cup W_j)| = 1$, then at most one set $W \in \mathcal{W}_0 - \{W_i, W_j\}$ satisfies $W \cap W_i \cap W_j \neq \emptyset$.
(v) If $|N_G(W_i \cup W_j)| = 2$, then for every $W \in \mathcal{W}_0 - \{W_i, W_j\}$ with $W_i \cap W_j \cap W \neq \emptyset$, we have $N_G(W_i \cup W_j) \cap W \cap S_0 \neq \emptyset$.

Proof. Omitted due to space limitation. □

Lemma 9. *$f(V, \mathcal{W}_0) \leq 4$ holds. In particular, for a vertex $v \in V$ whose frequency is four, the four distinct sets $W_i \in \mathcal{W}_0$, $i = 1, 2, 3, 4$ with $v \in W_i$ satisfy the following (4):*

$$\text{For some two sets } W_1, W_2,\ d(s_1) = 4 \text{ and } d(s_2) \geq 3 \text{ hold and any set in } \mathcal{W}_0 - \{W_1, W_2, W_3, W_4\} \text{ is disjoint with } W_1 \cup W_2. \tag{4}$$

Proof. Let W_1 and W_2 denote deficient sets in $\mathcal{W}_0$ with $W_1 \cap W_2 \neq \emptyset$. We observe how many sets in $\mathcal{W}_0 - \{W_1, W_2\}$ can intersect with $W_1 \cap W_2$. From Lemma 8(i)(ii), we have $|N_G(W_1 \cup W_2)| \geq 1$ and $|N_G(W_1 \cap W_2)| \geq 2$. Moreover, Lemma 8(iii) says that if $|N_G(W_1 \cap W_2)| = 2$, then every set $W \in \mathcal{W}_0 - \{W_1, W_2\}$ is disjoint with $W_1 \cap W_2$.

Consider the case where $|N_G(W_1 \cap W_2)| \geq 3$. By (2) and $|N_G(W)| \leq d^* - 1 \leq 3$ for each $W \in \mathcal{W}_0$, we have $|N_G(W_1 \cup W_2)| \leq 3$. In particular, if $|N_G(W_1 \cup W_2)| = 3$ (resp. $|N_G(W_1 \cup W_2)| = 2$), then we have $|N_G(W_1)| = |N_G(W_2)| = |N_G(W_1 \cap W_2)| = 3$ (resp. $|N_G(W_1)| = 3$ and $|N_G(W_2)| \geq 2$ without loss of generality). There are the following three possible cases (I) $|N_G(W_1 \cup W_2)| = 1$, (II) $|N_G(W_1 \cup W_2)| = 2$, $|N_G(W_1)| = 3$, and $|N_G(W_2)| \geq 2$, and (III) $|N_G(W_1 \cup W_2)| = 3$ and $|N_G(W_1)| = |N_G(W_2)| = |N_G(W_1 \cap W_2)| = 3$.

(I) Lemma 8(iv) implies that the frequency of each vertex in $W_1 \cap W_2$ is at most three.

(II) Assume that there are two distinct sets $W_3, W_4 \in \mathcal{W}_0 - \{W_1, W_2\}$ such that $W_1 \cap W_2 \cap W_3 \cap W_4 \neq \emptyset$. Lemma 8(v) implies that $N_G(W_1 \cup W_2) = \{s_3, s_4\}$. Hence, any other set $W \in \mathcal{W}_0$ cannot intersect with $W_1 \cup W_2$ by $W \cap \{s_3, s_4\} = \emptyset$ and the connectedness of $G[W]$. Therefore, we can observe that the frequency of each vertex in $W_1 \cap W_2$ is at most four and that if $W_1 \cap W_2 \cap W_3 \cap W_4 \neq \emptyset$ holds, then (4) holds.

(III) Assume that there is a set $W_3 \in \mathcal{W}_0 - \{W_1, W_2\}$ with $W_1 \cap W_2 \cap W_3 \neq \emptyset$. We also assume that $|N_G(W_3 \cup W_1)| = |N_G(W_2 \cup W_3)| = 3$ and hence $|N_G(W_3)| = 3$, since otherwise we can apply the above arguments. Note that $d(s_1) = d(s_2) = d(s_3) = 4$. Then we have the following claim whose proof is omitted due to space limitation. This claim proves this lemma.

Claim. Every set in $\mathcal{W}_0 - \{W_1, W_2, W_3\}$ is disjoint with $W_1 \cap W_2 \cap W_3$. □

Lemma 10. *Let W_i, W_j be two minimal deficient sets wrt. v_i and v_j, respectively, such that $W_i \cap W_j \neq \emptyset$, $|N_G(W_i \cup W_j)| \leq 2$, $d(v_i) = 4$, and $\{v_i, v_j\} \cap (W_i \cap W_j) = \emptyset$. Then for any feasible solution S to 4LVSLP, we have $|S \cap (W_i \cup W_j)| \geq 2$.*

Proof. By Lemma 2, we have $S \cap (W_i \cup W_j) \neq \emptyset$; let $s \in S \cap (W_i \cup W_j)$. If $s \neq v_i$, then $|N_G(W_i \cup W_j - \{s\})| \leq |N_G(W_i \cup W_j)| + 1 \leq 3 < d(v_i)$. Hence, in this case, again by Lemma 2, we have $S \cap (W_i \cup W_j - \{s\}) \neq \emptyset$ and $|S \cap (W_i \cup W_j)| \geq 2$. If $s = v_i$, then $s = v_i \notin W_j$ holds and hence by Lemma 2 we have $(S - s) \cap W_j \neq \emptyset$. Also in this case, $|S \cap (W_i \cup W_j)| \geq 2$. □

Lemma 11. *S_0 is 3-approximate.*

Proof. Let S^* denote an optimal solution. Since S^* is feasible, we have $W \cap S^* \neq \emptyset$ for every $W \in \mathcal{W}_0$. Consider a mapping $g : \mathcal{W}_0 \to S^*$ such that for each set $W \in \mathcal{W}_0$, $f(W) = s^*$ holds for some source $s^* \in S^*$ with $s^* \in W$. If $|\{W \in \mathcal{W}_0 \mid g(W) = s^*\}| \leq 3$ holds for each source $s^* \in S^*$, then we have $|\mathcal{W}_0| \leq 3|S^*|$, from which $|S_0| = |\mathcal{W}_0| \leq 3|S^*|$. We claim that there is such a mapping.

Assume that for a mapping g, there is a source $s_1^* \in S^*$ which at least four sets in $\mathcal{W}_0$ is mapped to. By Lemma 9, $f(V, \mathcal{W}_0) \leq 4$ holds, and hence the number of sets in $\mathcal{W}_0$ mapped to s_1^* is exactly four. Moreover, the four sets W_1, W_2, W_3, W_4 in $\mathcal{W}_0$ with $g(W_i) = s_1^*$, $i = 1, 2, 3, 4$ satisfy (4); $|N_G(W_1 \cup W_2)| = 2$, $d(s_1) = 4$, and $W \cap (W_1 \cup W_2) = \emptyset$ for each $W \in \mathcal{W}_0 - \{W_1, W_2, W_3, W_4\}$.

Now Lemma 10 implies that $W_1 \cup W_2$ includes a source $s_2^* \in S^* - \{s_1^*\}$. Notice that no set in $\mathcal{W}_0$ is mapped to s_2^* in g because every set $W \in \mathcal{W}_0 - \{W_1, W_2, W_3, W_4\}$ satisfies $s_2^* \notin W$ and each of W_i, $i = 1, 2, 3, 4$ has been mapped to s_1^*. So, we can decrease the number of sets in $\mathcal{W}_0$ mapped to s_1^* by one, by remapping one of two sets W_1 and W_2 including s_2^* to s_2^*. Consequently, by repeating this arguments, we can obtain a required mapping. □

We now give a tight example for the algorithm GREEDY_LVSLP. Let $H_i = (V_i, E_i)$ be the graph where $V_i = \bigcup_{j=1}^{3} \{v_j^i, u_{j1}^i, u_{j2}^i, u_{j3}^i\}$ and $E_i = (\bigcup_{j,\ell}(v_j^i, v_\ell^i)) \cup (\bigcup_{j=1}^{3} \{(u_{j1}^i, u_{j2}^i), (u_{j1}^i, u_{j3}^i), (u_{j1}^i, v_1^i), (u_{j1}^i, v_2^i), (u_{j1}^i, v_3^i)\})$ (see Fig. 3(a)). Let $G = (V, E)$ be the graph where $V = \{u_1, u_2, u_3\} \cup (\bigcup_{i=1}^{q} V_i)$, $q \geq 4$ and $E = \bigcup_{i=1}^{q}(E_i \cup (\bigcup_{j=1}^{3} \{(u_j, u_{j2}^i), (u_j, u_{j3}^i)\}))$, $d(u_{j1}^i) = 4$ for each $i \in \{1, 2, \ldots, q\}$ and $j \in \{1, 2, 3\}$, and $d(v) = 0$ for all other vertices (see Fig. 3(b)). For G and d, the algorithm GREEDY_LVSLP returns a source set $S_0 = \bigcup_{i=1}^{q} \{u_{11}^i, u_{21}^i, u_{31}^i\}$ and $\mathcal{W}_0 = \bigcup_{i=1}^{q} \{\{u_{11}^i, u_{12}^i, u_{13}^i, v_1^i, v_2^i, v_3^i\}, \{u_{21}^i, u_{22}^i, u_{23}^i, v_1^i, v_2^i, v_3^i\}, \{u_{31}^i, u_{32}^i, u_{33}^i, v_1^i, v_2^i, v_3^i\}\}$. On the other hand, $\{v_1^1, v_1^2, \ldots, v_1^q\}$ is an optimal solution. This

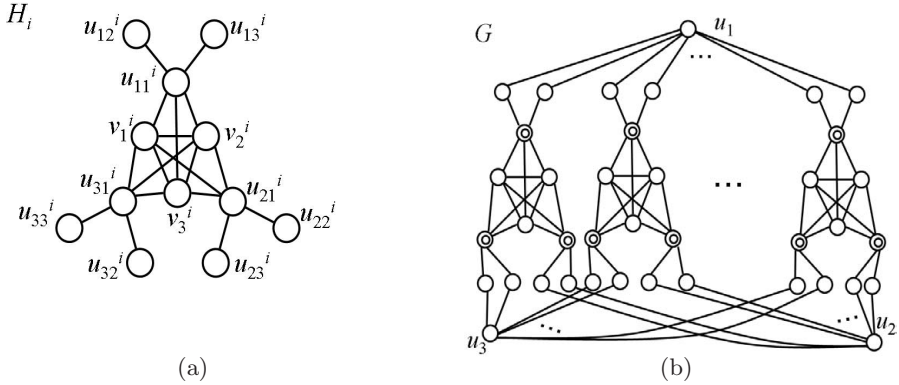


Fig. 3. Illustration of a tight example G for the algorithm GREEDY_LVSLP in the case of $d^* = 4$. (a) shows the graph H_i which is a subgraph of G in (b). In G, each vertex v with $d(v) = 4$ is drawn as double circles.

example shows that our analysis of the algorithm is tight. We here remark that in a similar way, we can construct an instance in which GREEDY_LVSLP returns a solution S with $|S| = (d^* - 1)opt(G, d)$ for a general d.

Finally, we show that the problem is APX-hard. In [7], it was shown that 4LVSLP is NP-hard by a reduction from the minimum vertex cover problem restricted to 3-regular graphs:

Vertex-cover problem in a 3-regular graph (VC3R)
INSTANCE: $(G = (V, E), k)$: A 3-regular graph $G = (V, E)$ and an integer k.
QUESTION: Is there a vertex cover X with $|X| \leq k$ in G? □

where a set $V' \subseteq V$ of vertices is called a *vertex cover* if every edge $e = (u, v) \in E$ satisfies $\{u, v\} \cap V' \neq \emptyset$, and a graph is called *k-regular* if the degree of every vertex is exactly k. As shown in [1], the minimum vertex cover problem is APX-hard, even restricted to 3-regular graphs. We can prove the APX-hardness of 4LVSLP by using the same reduction as [7].

Lemma 12. *4LVSLP is APX-hard.*

Proof. Omitted due to space limitation. □

5 Concluding Remarks

In this paper, given an undirected graph $G = (V, E)$ and a demand function $d : V \to Z^+$, we have considered the problem of finding a source set $S \subseteq V$ with the minimum cardinality for which there exist $d(v)$ paths between every vertex $v \in V - S$ and S such that no pair of paths has a common vertex in $V - v$. We have shown that a simple greedy algorithm finds a $\max\{d^* + 1, 2d^* - 6\}$-approximate solution to the problem in $O(\min\{d^*, \sqrt{n}\}d^*n^2)$ time. Especially, restricted to $d^* \leq 4$, we have given a tight analysis to show that it achieves an approximation ratio of 3, while the problem is APX-hard. However, it is still open whether the problem is approximable within a constant which is independent of d^*.

Acknowledgments. This research was partially supported by the Scientific Grant-in-Aid from Ministry of Education, Culture, Sports, Science and Technology of Japan.

References

[1] Alimonti, P., Kann, V.: Some APX-completeness results for cubic graphs. Theoretical Computer Science 237, 123–134 (2000)
[2] Arata, K., Iwata, S., Makino, K., Fujishige, S.: Locating sources to meet flow demands in undirected networks. Journal of Algorithms 42, 54–68 (2002)
[3] Bárász, M., Becker, J., Frank, A.: An algorithm for source location in directed graphs. Operations Research Letters 33(3), 221–230 (2005)
[4] Even, S., Tarjan, R.E.: Network flow and testing graph connectivity. SIAM Journal on Computing 4, 507–518 (1975)

[5] van den Heuvel, J., Johnson, M.: Transversals of subtree hypergraphs and the source location in digraphs. DAM Research Report LSE-CDAM-2004-10, London School of Economics (2004)
[6] Ishii, T., Fujita, H., Nagamochi, H.: Minimum cost source location problem with local 3-vertex-connectivity requirements. Theoretical Computer Science 372(1), 81–93 (2007)
[7] Ishii, T., Fujita, H., Nagamochi, H.: Source location problem with local 3-vertex-connectivity requirements. Discrete Applied Mathematics (to appear)
[8] Ito, H.: Telecommunication network simplifying problem – connectivity, area graph and T-mixed cut. NTT R&D 44(4), 367–372 (1995) (in Japanese)
[9] Ito, H., Ito, M., Itatsu, Y., Nakai, K., Uehara, H., Yokoyama, M.: Source location problems considering vertex-connectivity and edge-connectivity simultaneously. Networks 40(2), 63–70 (2002)
[10] Ito, H., Makino, K., Arata, K., Honami, S., Itatsu, Y., Fujishige, S.: Source location problem with flow requirements in directed networks. Optimization Methods and Software 18, 427–435 (2003)
[11] Labbe, M., Peeters, D., Thisse, J.-F.: Location on networks. In: Ball, M.O., et al. (eds.) Handbooks in Operations Research and Management Science, vol. 8, pp. 551–624. North Holland, Amsterdam (1995)
[12] Nagamochi, H., Ibaraki, T.: A linear-time algorithm for finding a sparse k-connected spanning subgraph of a k-connected graph. Algorithmica 7, 583–596 (1992)
[13] Nagamochi, H., Ishii, T., Ito, H.: Minimum cost source location problem with vertex-connectivity requirements in digraphs. Information Processing Letters 80(6), 287–294 (2001)
[14] Sakashita, M., Makino, K., Fujishige, S.: Minimum cost source location problems with flow requirements. In: Proceedings of the 7th Latin American Theoretical Informatics, pp. 769–780 (2006)
[15] Tamura, H., Sengoku, M., Shinoda, S., Abe, T.: Location problems on undirected flow networks. IEICE Transactions E73-E(12), 1989–1993 (1990)
[16] Tamura, H., Sugawara, H., Sengoku, M., Shinoda, S.: Plural cover problem on undirected flow networks. IEICE Transactions J81-A, 863–869 (1998) (in Japanese)

Dynamic Distance Hereditary Graphs Using Split Decomposition*

Emeric Gioan and Christophe Paul**

CNRS - LIRMM, Université de Montpellier 2, France

Abstract. The problem of maintaining a representation of a dynamic graph as long as a certain property is satisfied has recently been considered for a number of properties. This paper presents an optimal algorithm for this problem on vertex-dynamic connected distance hereditary graphs: both vertex insertion and deletion have complexity $O(d)$, where d is the degree of the vertex involved in the modification. Our vertex-dynamic algorithm is competitive with the existing linear time recognition algorithms of distance hereditary graphs, and is also simpler. Besides, we get a constant time edge-dynamic recognition algorithm. To achieve this, we revisit the split decomposition by introducing graph-labelled trees. Doing so, we are also able to derive an intersection model for distance hereditary graphs, which answers an open problem.

1 Introduction

Motivated by their practical applications as well as by their related theoretical challenges, dynamic graph algorithms have received particular attention over the last few years [12]. Solving a problem on a dynamic graph consists of an algorithm that, under a series of graph modifications (vertex or edge modification), updates a data structure supporting elementary queries (e.g. adjacency). Let us note that the series of modifications to which the graph is submitted is not known in advance. To be of interest, such an algorithm should not recompute a solution from scratch. In order to ensure locality of the computation, most of the known dynamic graph algorithms are based on decomposition techniques. For example, the SPQR-tree data structure has been introduced in order to dynamically maintain the 3-connected components of a graph which allows on-line planarity testing [2].

This paper considers the *dynamic representation problem* which asks for the maintenance of a representation of a dynamic graph as long as a certain property Π is satisfied. Existing literature on this problem includes representation of chordal graphs [18], proper interval graphs [16], cographs [21], directed cographs [7], permutation graphs [8]. The data structures used for the last four results are strongly related to the modular decomposition tree [14]. The *split decomposition* (also called 1-join decomposition), introduced by Cunningham [9],

* Research supported by the French ANR project "*Graph Decompositions and Algorithms* (GRAAL)".

** Research conducted while C. Paul was on Sabbatical at School of Computer Science, McGill University, Montréal, Canada.

T. Tokuyama (Ed.): ISAAC 2007, LNCS 4835, pp. 41–51, 2007.

is a generalization of the modular decomposition. A natural question is to ask whether the split decomposition can be used to dynamically represent wider graph families? We answer positively to this question.

The algorithmic aspects of the split decomposition, unlike the modular decomposition, are not well understood. For instance, totally decomposable graphs are known to be the *distance hereditary graphs* [1,15], which form an interesting family of graphs for several reasons: they generalize the well-known cographs [5], which are totally decomposable by the modular decomposition; they are the graphs of rankwidth 1 [20] and are among the elementary graphs of clique-width 3 [4]; they also have various theoretical characterizations... Computing the split decomposition in linear time [10] is very complicated. It follows that most of the known algorithms (even recent ones) operating on distance hereditary graphs do not rely on the split decomposition but rather on a heavy breadth-first search layering characterization [1], or on some ad-hoc (rooted) tree decompositions [3,17,23]. Similarly the recent $O(1)$ edge-only dynamic recognition algorithm [6] is based on the BFS layering characterization.

In this paper, we revisit the split decomposition theory [9] under the new framework of *graph-labelled trees*, which formalize the (unrooted) tree decomposition underlying the split decomposition (see Section 2). As a by-product, we can derive two new characterizations of distance hereditary graphs (see Section 3). The first one is an intersection model for the family of distance hereditary graphs. The existence of such a model was known [19], but to our knowledge, the model itself was still not discovered [22]. Graph-labelled trees also yield an incremental characterization of distance hereditary graphs from which we can deduce an optimal vertex-dynamic algorithm to represent distance hereditary graphs (see Section 4). These are the first results obtained for the split decomposition in the framework of graph-labelled trees. Moreover the simplicity of the solutions we propose witnesses the elegance of graph-labelled trees: e.g. our $O(d)$ vertex insertion algorithm is not only competitive with the existing static linear time recognition algorithms [3,11,15], but is also simpler. We also get a constant time edge-dynamic recognition algorithm, different from [6] (see Section 5). We believe that new applications or generalizations of the split decomposition could be derived from graph-labelled trees.

2 Split Decomposition and Graph-Labelled Trees

Any graph $G = (V(G), E(G))$ we consider is simple and loopless. For a subset $S \subseteq V(G)$, $G[S]$ is the subgraph of G induced by S. If T is a tree and S a subset of leaves of T, then $T(S)$ is the smallest subtree of T spanning the leaves of S. If x is a vertex of G then $G - x = G[V(G) - \{x\}]$. Similarly if $x \notin V(G)$, $G + (x, S)$ is the graph G augmented by the new vertex x adjacent to $S \subseteq V(G)$. We denote $N(x)$ the neighbourhood of a vertex x. The neighborhood of a set $S \subseteq V(G)$ is $N(S) = \{x \notin S \mid \exists y \in S,\ xy \in E(G)\}$. The *clique* is the complete graph and the star is the complete bipartite graph $K_{1,n}$. The universal vertex of the star is called its *centre* and the degree one vertices its *extremities*.

Definition 2.1. [9] *A* split *of a graph G is a bipartition (V_1, V_2) of $V(G)$ such that 1) $|V_1| \geqslant 2$ and $|V_2| \geqslant 2$; and 2) every vertex of $N(V_1)$ is adjacent to every vertex of $N(V_2)$.*

Cliques and stars are called *degenerate* since for any such graph on at least 4 vertex, any non-trivial vertex bipartition is a split. A graph with no split that is not degenerate is called *prime*. The split decomposition of a graph G, as originally studied in [9], consists of: finding a split (V_1, V_2), decomposing G into $G_1 = G[V_1 \cup \{x_1\}]$, with $x_1 \in N(V_1)$ and $G_2 = G[V_2 \cup \{x_2\}]$ with $x_2 \in N(V_2)$, x_1 and x_2 being called the *marker vertices*; and then recursing on G_1 and G_2. In [9], Cunningham presents the idea of a tree decomposition. But its main result stating the uniqueness of lastly resulting graphs in a split decomposition focuses on the set of resulting graphs more than on the structure linking them together. To reformulate Cunningham's result in terms of tree, let us introduce some terminology.

Definition 2.2. *A* graph-labelled tree $(T, \mathcal{F})$ *is a tree in which any node v of degree k is labelled by a graph $G_v \in \mathcal{F}$ on k vertices such that there is a bijection ρ_v from the tree-edges incident to v to the vertices of G_v.*

We call *nodes* the internal vertices of a tree, and *leaves* the other ones. Let $(T, \mathcal{F})$ be a graph-labelled tree and l be a leaf of T. A node or a leaf u different from l is *l-accessible* if for any tree-edges $e = wv$ and $e' = vw'$ on the l, u-path in T, we have $\rho_v(e)\rho_v(e') \in E(G_v)$. By convention, the unique neighbor of l in T is also l-accessible. See Figure 1 for an example.

Definition 2.3. *The* accessibility graph $G(T, \mathcal{F})$ *of a* graph-labelled tree $(T, \mathcal{F})$ *has the leaves of T as vertices, and there is an edge between x and y if and only if y is x-accessible.*

Lemma 2.1. *Let $(T, \mathcal{F})$ be a graph-labelled tree and T_1, T_2 be the subtrees of $T - e$ where e is a tree-edge non-incident to a leaf. Then the bipartition (L_1, L_2) of the leaves of T, with L_i being the leaf set of T_i, defines a split in the graph $G(T, \mathcal{F})$.*

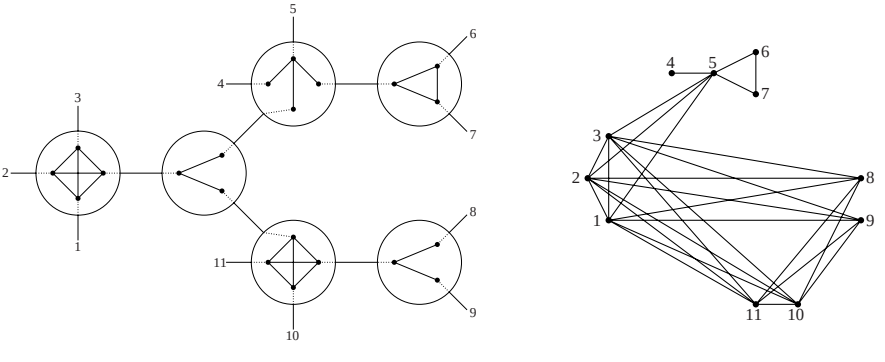


Fig. 1. A graph-labelled tree and its accessibility graph

We can naturally define the *split* and the converse *join* operations on a graph-labelled tree $(T, \mathcal{F})$ as follows (see Figure 2):

- *Split of* $(T, \mathcal{F})$*:* Let v be a node of T whose graph G_v has a split (A, B). Let G_A and G_B be the subgraphs resulting from the split (A, B) and a, b be the respective marker vertices. Splitting the node v consists in substituting v by two adjacent nodes v_A and v_B respectively labelled by G_A and G_B such that for any $x \in V(G_A)$ different from a, $\rho_{v_A}(x) = \rho_v(x)$ and $\rho_{v_A}(a) = v_A v_B$ (similarly for any $x \in V(G_B)$ different from b, $\rho_{v_B}(x) = \rho_v(x)$ and $\rho_{v_B}(b) = v_A v_B$).
- *Join of* $(T, \mathcal{F})$*:* Let uv be a tree-edge of T. Then joining the nodes u and v consists in substituting them by a single node w labelled by G_w the accessibility graph of the tree with only edge u, v and respective labels G_u and G_v. For any vertex x of G_w, $\rho_w(x) = \rho_v(x)$ or $\rho_w(x) = \rho_u(x)$ depending on which graph x belonged.

Observe that if $(T, \mathcal{F})$ is obtained from $(T', \mathcal{F}')$ by a join or a split operation, then it follows from the definitions that $G(T, \mathcal{F}) = G(T', \mathcal{F}')$.

Among the join operations, let us distinguish: the *clique-join*, operating on two neighboring nodes labelled by cliques, and the *star-join*, operating on star-labelled neighboring nodes u, v such that the tree-edge uv links the centre of one star to an extremity of the other. A graph-labelled tree $(T, \mathcal{F})$ is *reduced* if neither clique-join nor star-join can be applied, i.e. the clique nodes are pairwise non-adjacent and two star nodes u and v can be adjacent only if $\rho_u(uv)$ and $\rho_v(uv)$ are both centres or both extremities of their respective stars G_u and G_v.

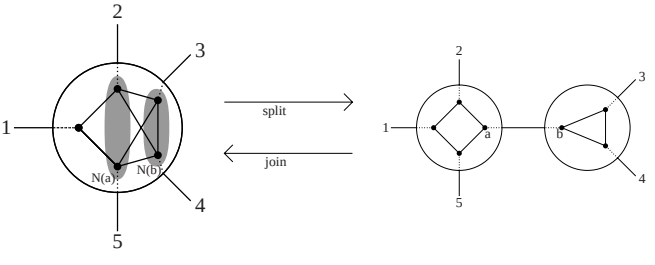


Fig. 2. The split and the join operations on a graph-labelled tree

Theorem 2.1 (Cunningham's Theorem reformulated). *For any connected graph G, there exists a unique reduced graph-labelled tree $(T, \mathcal{F})$ such that $G = G(T, \mathcal{F})$ and any graph of $\mathcal{F}$ is prime or degenerate.*

For a connected graph G, the *split tree* $ST(G)$ of G is the unique reduced graph-labelled tree $(T, \mathcal{F})$ in the above Theorem 2.1 (see Figure 1).

The next two lemmas are central to proofs of further theorems.

Lemma 2.2. *Let $ST(G) = (T, \mathcal{F})$ be the split tree of a connected graph G. Let l be a leaf of T, and $e = uv$, $e' = uv'$ be distinct tree edges such that u is a*

l-accessible and e is on the u, l path in T. Then $\rho_u(e)\rho_u(e') \in E(G_u)$ if and only if there exists a l-accessible leaf l' in the subtree of $T - e'$ containing v'.

The above easy lemma can be rephrased as follows: if u and v are two adjacent l-accessible nodes, then there exists a l-accessible leaf l' such that the l, l'-path contains the tree edge u, v. It follows that:

Lemma 2.3. *Let $ST(G) = (T, \mathcal{F})$ be the split tree of a connected graph G. For any vertex $x \in V(G)$, $T(N(x))$ has at most $2.|N(x)|$ nodes.*

3 Characterizations of Distance Hereditary Graphs

A graph is *distance hereditary* (DH for short) if the distance between any given pair of vertices remains the same in any connected induced subgraphs. By [15], a graph is DH if and only if it is totally decomposable by the split decomposition, i.e. its split tree is labelled by cliques and stars. Hence DH graphs are exactly accessibility graphs of clique-star labelled trees, *clique-star trees* for short. Among the possible clique-star trees, the split tree is the unique reduced one. Figure 1 gives an example. We mention that ternary clique-star trees were used in [13] to draw DH graphs. Another general example is given by cographs, which can be characterized as distance hereditary graphs whose stars in the split tree are all directed towards a root of the tree.

An intersection model. Given a family S of sets, one can define the intersection graph $\mathcal{I}(S)$ as the graph whose vertices are the elements of S and there is an edge between two elements if and only if they intersect. Many restricted graph families are defined or characterized as the intersection graphs (e.g. chordal graphs, interval graphs... see [19]). Graph families supporting an intersection model can be characterized without even specifying the model [19]. This result applies to DH graphs, and no model was known [22]. Based on clique-star trees, an intersection model can be easily derived. Note that it can be equivalently stated by considering only reduced clique-star trees, or even only ternary ones. We call *accessibility set* of a leaf l in a graph-labelled tree the set of pairs $\{l, l'\}$ with l' a l-accessible leaf.

Theorem 3.1 (Intersection model). *A graph is distance hereditary if and only if it is the intersection graph of a family of accessibility sets of leaves in a set of clique-star trees.*

Incremental characterization. Let G be a connected DH graph and let $ST(G) = (T, \mathcal{F})$ be its split tree. Given a subset S of $V(G)$ and $x \notin V(G)$, we want to know whether the graph $G + (x, S)$ is DH or not.

Definition 3.1. *Let $T(S)$ be the smallest subtree of T with leaves S. Let u be a node of $T(S)$.*

1. *u is* fully-accessible *if any subtree of $T - u$ contains a leaf $l \in S$;*

2. *u is* singly-accessible *if it is a star-node and exactly two subtrees of $T - u$ contain a leaf $l \in S$ among which the subtree containing the neighbor v of u such that $\rho_u(uv)$ is the centre of G_u;*
3. *u is* partially-accessible *otherwise.*

We say that a star node is *oriented towards* an edge (or a node) of T if the tree-edge mapped to the centre of the star is on the path between the edge (or node) and the star.

Theorem 3.2 (Incremental characterization). *Let G be a connected distance hereditary graph and $ST(G) = (T, \mathcal{F})$ be its split tree. Then $G + (x, S)$ is distance hereditary if and only if:*

1. *at most one node of $T(S)$ is partially accessible;*
2. *any clique node of $T(S)$ is either fully or partially-accessible.*
3. *if there exists a partially accessible node u, then any star node $v \neq u$ of $T(S)$ is oriented towards u if and only if it is fully accessible. Otherwise, there exists a tree-edge e of $T(S)$ towards which any star node of $T(S)$ is oriented if and only if it is fully accessible.*

4 A Vertex-Only Fully-Dynamic Recognition Algorithm

In this section we mainly compute the characterization given by Theorem 3.2 to obtain the following main result.

Main Theorem 4.1. *There exists a fully dynamic recognition algorithm for connected distance hereditary graphs with complexity $O(d)$ per vertex insertion or deletion operation involving d edges.*

The vertex insertion algorithm yields a linear time recognition algorithm of (static) DH graphs, thereby achieving the best known bound but also simplifying the previous non-incremental ones [15,11,3].

Corollary 4.1 (Static recognition). *The vertex insertion routine enables to recognize distance hereditary graphs in linear time.*

The following data structure is used for the clique-star tree $ST(G) = (T, \mathcal{F})$ of the given DH graph G: a (rooted) representation of the tree T; a single *clique-star marker* distinguishing the type of each node; a *centre-marker* distinguishing the centre of each star node; and the degree of each node. Let us notice that this data structure is an $O(n)$ space representation of the DH graphs on n vertices.

4.1 Vertex Insertion Algorithm

Computing the smallest subtree spanning a set of leaves. Given a set S of leaves of a tree T, we need to identify the smallest subtree $T(S)$ spanning S, and to store the degrees of its nodes. This problem is easy to solve on rooted (or directed) trees by a simple bottom-up marking process with complexity $O(|T(S)|)$. So an arbitrary root of T is fixed.

1. Mark each leaf of S. A node is *active* if its father is not marked.
2. Each marked node marked its father as long as: 1) the root is not marked and there is more than one active node, or 2) the root is marked and there is at least one active node.
3. As long as the root of the subtree induced by the marked nodes is a leaf not in S, remove this node and check again.

Testing conditions of Theorem 3.2. The first two conditions of Theorem 3.2 are fairly easy to check by following Definition 3.1: a node u is fully-accessible if its degrees in $T(S)$ and T are the same; u is singly-accessible if it is a star, if it has degree 2 in $T(S)$ and if the centre neighbor belongs to $T(S)$; and u is partially accessible otherwise, such a node having to be unique if it exists. This test costs $O(|T(S)|)$.

We now assume that the first two conditions of Theorem 3.2 are fulfilled. At first, the case is trivial if $|S| = 1$. So assume $|S| > 1$.

We define *local orientations* on vertices of a tree as the choice, for every vertex u, of a vertex $f(u)$ such that either $f(u) = u$ or $f(u)$ is a neighbor of u. Local orientations are called *compatible* if 1) $f(u) = u$ implies $f(v) = u$ for every neighbor v of u, and 2) $f(u) = v$ implies $f(w) = u$ for every neighbor $w \neq v$ of u. It is an easy exercise to see that if local orientations are compatible then exactly one of the two following properties is true: either there exists a unique vertex u with $f(u) = u$, in this case u is called *node-root*, or there exists a unique tree-edge uv with $f(u) = v$ and $f(v) = u$, in this case uv is called *edge-root*.

The test for the third condition of Theorem 3.2 consists of building, if possible, suitable compatible local orientations in the tree $T(S)$:

1. Let u be a leaf of $T(S)$. Then $f(u)$ is the unique neighbor of u.
2. Let u be a star node of $T(S)$. If u is partially-accessible, then $f(u) = u$. If u is singly-accessible, then $f(u) = v$ with v the unique neighbor v of u belonging to $T(S)$ such that $\rho_u(uv)$ is an extremity of the star. If u is fully-accessible, then $f(u) = v$ with v the neighbor of u such that $\rho_u(uv)$ is the centre of the star.
3. Let u be a clique node of $T(S)$. If u is partially-accessible, then $f(u) = u$. Otherwise, u is fully-accessible and its neighbors are leaves or star nodes. If $f(v) = u$ for every neighbor v of u then $f(u) = u$. If $f(v) = u$ for every neighbor v of u but one, say w, then $f(u) = w$. Otherwise u is an *obstruction*.

The third condition of Theorem 3.2 is satisfied if and only if 1) there is no obstruction and 2) local orientations of $T(S)$ are compatible. This test can be done in time $O(|T(S)|)$ by a search of $T(S)$. Hence, the conditions of Theorem 3.2 can be tested in $O(|T(S)|)$ time.

Updating the split-tree. We now assume that graph $G + (x, S)$ is DH. So by Theorem 3.2 the split tree has either a unique node-root or a unique edge-root. There are three cases (see Figure 3).

1. There is a node-root u being partially-accessible, or S is reduced to a unique vertex u. We may have to make a first update of T by splitting the node u

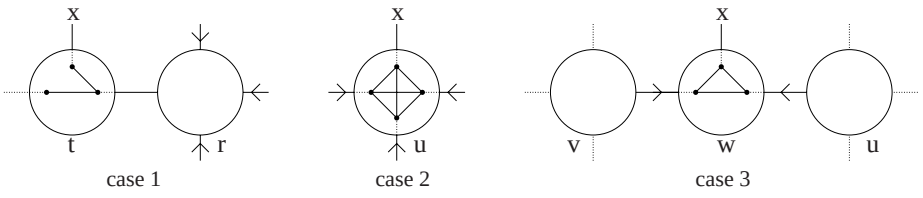


Fig. 3. Vertex insertion

under some conditions on the degrees. Let U, resp. A, be the set of tree-edges adjacent to u in T, resp. in $T(S)$.

(a) If u is a clique node with $|U \setminus A| \geq 2$, then u is replaced with an edge vw in T. Then v, resp. w, is labelled by a clique whose vertices correspond to A, resp. $U \setminus A$, except one which corresponds to vw. In this case, v is now the node-root.

(b) If u is a star node with centre mapped to the tree edge e and $|(U \setminus A) \setminus e)| \geq 1$, then u is replaced with an edge vw in T. Then v is labelled by a star whose extremities correspond to $A \setminus \{e\}$ and centre to vw (we have $|A \setminus \{e\}| > 1$ since u is not singly accessible), and w is labelled by a star whose extremities correspond to $(U \setminus A) \cup \{vw\}$ and centre to e. If $e \in A$, then the edge vw is now the edge-root. And if $e \notin A$, then the node v becomes the node-root.

If the tree still has a node-root $r = u$ or $r = v$, then let s be its neigbhor in T that does not belong to $T(S)$. Then the insertion edge is $e = rs$, and $ST(G + (x, S))$ is obtained by subdividing rs with a star node t of degree 3 whose centre is $\rho_w(rt)$ and making the leaf x adjacent to t. Then, in the case where s is a star with centre $\rho_v(st)$, we proceed a join operation on the tree-edge st .

2. There is a node-root u not being partially-accessible, Then u is a clique node, and $ST(G+(x, S))$ is obtained by adding the new leaf x adjacent to u whose degree thereby increases by one.
3. There is an edge-root uv. Then $ST(G + (x, S))$ is obtained by subdividing uv with a clique node w of degree 3 and making the leaf x adjacent to w.

Each above update operation can be done in $O(1)$ time, except the splitting in case 1 which requires $O(|T(S)|)$ time (by deleting A from u to get w, and adding A to a new empty node v). Any other update operation requires $O(1)$ time to maintain the data structure of the split tree (artificial root, degrees...). Then, the complexity for the whole insertion algorithm derives from from previous steps and the fact that $O(|T(S)|) = O(|S|)$ (Lemma 2.3).

Theorem 4.2 (Vertex insertion). *Let $G + (x, S)$ be a graph such that G is a connected distance hereditary graph. Given the data structure of split tree $ST(G)$, testing whether $G + (x, S)$ is distance hereditary and if so computing the data structure of $ST(G + (x, S))$ can be done in $O(|S|)$ time.*

4.2 Vertex Deletion Algorithm

Removing a vertex x from a distance hereditary graph G always yields a distance hereditary graph $G - x$. Let $ST(G)$ be the split tree of G. Updating the data structure of the split tree can be done as follows.

1. Remove the leaf x and update the degree of its neighbor v.
2. If v now has degree 2, then replace v with an edge between its neighbors, and, if needed, reduce the clique-star tree on this edge.
3. If v is a star node whose centre neighbor was x, then $G - x$ is no longer connected, and the split-trees of each connected component are the components of $T - \{v, x\}$.

It is easy to see that any operation costs $O(1)$ except the join operation which costs $min(d', d'')$ where d', d'' are degree of the concerned nodes. Since at least one of these nodes is fully accessible, this minimum degree is lower than d. Hence this join operation costs $O(d)$.

Lemma 4.1 (Vertex deletion). *Let G be a connected distance hereditary graph and x be a degree d vertex of G. Given the data structure of split tree $ST(G)$, testing whether $G - x$ is a connected distance hereditary graph and if so computing the data structure of $ST(G - x)$ can be done in $O(d)$ time.*

5 Other Applications and Concluding Remarks

The results in this Section, and the previous ones, will be fully developped, with proofs (omitted here for lack of space), in a forthcoming paper.

A constant time edge-only fully-dynamic recognition algorithm. We consider the problem of adding or deleting an edge to a connected distance hereditary graph and testing if the new graph is distance hereditary. We use again the split tree of the graph to do this test easily and maintain the split tree in constant time. Another constant time algorithm for this problen, with other technique, has been developped in [6].

Let G be a connected distance hereditary graph, and x and y two vertices of G. If $xy \notin E(G)$ resp. $xy \in E(G)$, then let $G' = G + e$ resp. $G' = G - e$ with $e = xy$. Let P be the path from x to y in $ST(G)$, which defines a word W on the alphabet $\{K, L, R, S\}$, where K stands for a clique node, R for a star node with center directed towards x, L for a star node with center directed towards y, and S for a star node with center not directed towards x or y. Note that $xy \in E(G)$ if and only if W contains no letter S.

Theorem 5.1. *The graph G' is distance hereditary if and only if W has one of the following forms, where letters in brackets can be deleted.*
Case $G' = G + e$: $(R)SS(L)$, $(R)SK(L)$, $(R)KS(L)$, $(R)S(L)$.
Case $G' = G - e$: $(R)LR(L)$, $(R)LK(L)$, $(R)KR(L)$, $(R)K(L)$, $(R)(L)$.
In the deletion case, if $W = (R)(L)$, then G' is no longer connected.

As a consequence, we get the following constant time algorithm:

1. Test if W has length at most 4 and satisfies conditions of Theorem 5.1.
2. Update the split tree. Nodes of letters in brackets are called *extreme*.
   (a) If the not-extreme nodes are not ternary, then make a split on these nodes to get ternary nodes instead, in the path from x to y.
   (b) Replace the not-extreme nodes with ternary nodes according to the following table, and, in the cases where there are two not-extreme nodes, exchange the edges of the graph-labelled tree adjacent to these nodes, which are not in the path from x to y, between these nodes. Extreme nodes are unchanged.

edge insertion $\longrightarrow$ $\longleftarrow$ edge deletion	
$(R)SS(L)$	$(R)LR(L)$
$(R)SK(L)$	$(R)LK(L)$
$(R)KS(L)$	$(R)KR(L)$
$(R)S(L)$	$(R)K(L)$

   (c) If necessary, make (at most two) join operations involving the nodes that have been changed to get a reduced graph-labelled tree.

Particular case of cographs. We mention that algorithms in this paper can be adaptated to the particular case of cographs, which are totally decomposable for the modular decomposition. Indeed, a connected cograph is a connected distance hereditary graph of which split tree has the property that every star is directed towards a same given edge. The previous known fully-dynamic recognition algorithms of cographs (see [5] for the vertex insertion and [21] for the other operations) can be restated in the framework of graph-labelled trees, and then turn out to be equivalent to special cases of the previous algorithms.

General split decomposition. We finally mention that a generalization of our insertion algorithm to arbitrary graphs would have a complexity no longer linear in the number of edges. Consider the example where a new vertex is attached to the extermities of a path on $n \geqslant 4$ vertices (whose nodes in the split tree form a path of ternary stars). The resulting cycle is prime, witnessing $\Omega(n)$ changes in the split-tree representation. So, even for circle graphs, such an algorithm would have $\Omega(n)$ time complexity.

References

1. Bandelt, H.-J., Mulder, H.M.: Distance hereditary graphs. J. Combin. Theory Ser. B 41, 182–208 (1986)
2. Di Battista, G., Tamassia, R.: On-line planarity testing. SIAM J. Comput. 25(5), 956–997 (1996)
3. Bretscher, A.: LexBFS based recognition algorithms for cographs and related families. PhD thesis, Dep. of Computer Science, University of Toronto (2005)

4. Corneil, D., Habib, M., Lanlignel, J.-M., Reed, B., Rotics, U.: Polynomial Time Recognition of Clique-Width ≤ 3 Graphs. In: Gonnet, G.H., Viola, A. (eds.) LATIN 2000. LNCS, vol. 1776, pp. 126–134. Springer, Heidelberg (2000)
5. Corneil, D., Perl, Y., Stewart, L.K.: A linear time recognition algorithms for cographs. SIAM J. Comput. 14(4), 926–934 (1985)
6. Corneil, D., Tedder, M.: An optimal, edges-only fully dynamic algorithm for distance-hereditary graphs. In: Thomas, W., Weil, P. (eds.) STACS 2007. LNCS, vol. 4393, Springer, Heidelberg (2007)
7. Crespelle, C., Paul, C.: Fully-dynamic recognition algorithm and certificate for directed cographs. In: Hromkovič, J., Nagl, M., Westfechtel, B. (eds.) WG 2004. LNCS, vol. 3353, pp. 93–104. Springer, Heidelberg (2004) Discrete Appl. Math. 154(12), 1722–1741 (2006)
8. Crespelle, C., Paul, C.: Fully dynamic algorithm for recognition and modular decomposition of permutation graphs. In: Kratsch, D. (ed.) WG 2005. LNCS, vol. 3787, pp. 38–48. Springer, Heidelberg (2005)
9. Cunningham, W.H.: Decomposition of directed graphs. SIAM J. Alg. Disc. Meth. 3, 214–228 (1982)
10. Dahlhaus, E.: Parallel Algorithms for Hierarchical Clustering and Applications to Split Decomposition and Parity Graph Recognition. Journal of Algorithms 36(2), 205–240 (2000)
11. Damiand, G., Habib, M., Paul, C.: A simple paradigm for graph recognition: application to cographs and distance hereditary graphs. Th. Comp. Sci. 263, 99–111 (2001)
12. Eppstein, D., Galil, Z., Italiano, G.F.: Dynamic graph algorithms. In: Algorithms and Theory of Computation Handbook, ch. 8, CRC Press (1999)
13. Eppstein, D., Goodrich, M.T., Yu Meng, J.: Delta-confluent drawings. In: Healy, P., Nikolov, N.S. (eds.) GD 2005. LNCS, vol. 3843, pp. 165–176. Springer, Heidelberg (2006)
14. Gallai, T.: Transitiv orientierbare graphen. Acta Mathematica Acad. Sci. Hungar. 18, 25–66 (1967)
15. Hammer, P., Maffray, F.: Completely separable graphs. Discrete Appl. Math. 27, 85–99 (1990)
16. Hell, P., Shamir, R., Sharan, R.: A fully dynamic algorithm for recognizing and representing proper interval graphs. SIAM J. Comput. 31(1), 289–305 (2002)
17. Hseih, S.-Y., Ho, C.-W., Hsu, T.-S., Ho, M.-T.: Efficient algorithms for the hamiltonian problem on distance hereditary graphs. In: Ibarra, O.H., Zhang, L. (eds.) COCOON 2002. LNCS, vol. 2387, pp. 77–86. Springer, Heidelberg (2002)
18. Ibarra, L.: Fully dynamic algorithms for chordal graphs. In: Proc. 10th ACM-SIAM Annual Symp. on Disc. Algorithm (SODA), pp. 923–924 (1999)
19. McKee, T., McMorris, F.R.: Topics in intersection graph theory. SIAM Monographs on Discrete Mathematics and Applications 2 (1999)
20. Oum, S.I.: Rank-width and vertex-minors. J. Combin. Th. Ser. B 95, 79–100 (2005)
21. Shamir, R., Sharan, R.: A fully dynamic algorithm for modular decomposition and recognition of cographs. Discrete Appl. Math. 136(2-3), 329–340 (2004)
22. Spinrad, J.: List of open problems. `www.vuse.vanderbilt.edu/~spin/open.html`
23. Uehara, R., Uno, Y.: Canonical tree representation of distance hereditary graphs and its applications. IEICE Technical Report COMP2005-61 (2006)

Unifying Two Graph Decompositions with Modular Decomposition*

Binh-Minh Bui-Xuan[1], Michel Habib[2], Vincent Limouzy[2], and Fabien de Montgolfier[2]

[1] LIRMM - Univ. Montpellier II
buixuan@lirmm.fr
[2] LIAFA - Univ. Paris Diderot
{habib,limouzy,fm}@liafa.jussieu.fr

Abstract. We introduces the *umodules*, a generalization of the notion of graph module. The theory we develop captures among others undirected graphs, tournaments, digraphs, and 2−structures. We show that, under some axioms, a unique decomposition tree exists for umodules. Polynomial-time algorithms are provided for: non-trivial umodule test, maximal umodule computation, and decomposition tree computation when the tree exists. Our results unify many known decomposition like modular and bi-join decomposition of graphs, and a new decomposition of tournaments.

1 Introduction

In graph theory modular decomposition is now a well-studied notion [15,6,21,11], as well as some of its generalizations [10,19,23]. As having been rediscovered in other fields, the notion also appears under various names, including intervals, externally related sets, autonomous sets, partitive sets, and clans. Direct applications of modular decomposition include tractable constraint satisfaction problems, computational biology, graph clustering for network analysis, and graph drawing.

Besides, in the area of social networks, several vertex partitioning have been introduced in order to catch the idea of putting in the same part vertices acknowledging similar behaviour, in other words finding regularities [27]. Modular decomposition provides such a partitioning, yet seemingly too restrictive for real life applications. The concept of a role [12] on the other hand seems promising, however its computation unfortunately is $NP-$hard [13]. As a natural consequence, there is need for the search of *relaxed*, but *tractable*, variations of the modular decomposition scheme. A step following this direction has generalized graph modules to those of larger combinatorial structures, so-called homogeneous relations [3,4]. This paper follows the same research stream, and weakens

* Research supported by the ANR project *Graph Decompositions and Algorithms* (GRAAL) and by INRIA project-team Gang.

T. Tokuyama (Ed.): ISAAC 2007, LNCS 4835, pp. 52–64, 2007.

the definition of module in order to further decompose. Fortunately we obtain a new tractable variation of modular decomposition, that we now introduce.

Modular decomposition is based on *modules*, a vertex subset with no *splitter*. In graphs, a splitter of a vertex subset is linked with some, but not all, vertices of this subset. We shall see how this definition can be extended to homogeneous relations. The "outside" of a module constitutes therefore, for all vertices of the module, the same ordered partition. For instance, all vertices of an undirected graph module have the same neighbourhood. We here address unordered-modules, so-called *umodules* for short: the outside of a umodule constitutes for all vertices of the umodule the same unordered partition. For graph, the umodules are the *bijoins* (see Section 6). As there are clearly more umodules than modules, this allows deeper decomposition. We shall see that this decomposition is tractable.

After comparing umodule to previous notions in the topic, we display its tractability by giving an $O(|X|^4 \log |X|)$ time computation of the maximal umodules of a given homogeneous relation over a finite set X, and show how this can also be used as a non-trivial umodule existence test. The structure of the family of umodules is then investigated under different scenarios. We focus on a particular case, and provide a potent tractability theorem which makes use of the so-called *Seidel-switching* graph operation [26]. Fortunately enough, undirected graphs and tournaments fit into the latter formalism. We then deepen the study and address total decomposability issues, namely when any "large enough" substructure is decomposable. Surprisingly enough, this shows how our theory provides a very natural manner to obtain several results on *round tournaments*, including characterisation, recognition, and isomorphism testing (see e.g. [1] for more detailed information), as well as further computational results, such as the *feedback vertex set* computation.

2 Umodule, an Enlarged Notion of Module

A *diverse* triple of a finite set X is $(x, y, z) \subseteq X^3$ with $x \neq y$ and $x \neq z$, which will be denoted by $(x|yz)$ instead of (x, y, z) since the first element plays a particular role. Let H be a boolean relation over the diverse triples of X. Then, H_x denotes the binary relation on $X \setminus \{x\}$ such that $H_x(y, z) \Leftrightarrow H(x|yz)$.

Definition 1 (Homogeneous Relation and Module). *[3,4] H is a* homogeneous relation on X *if, for all $x \in X$, H_x is an equivalence relation on $X \setminus \{x\}$. A subset $M \subseteq X$ is a* module *of H if $H(x|mm')$ for all $m, m' \in M$ and $x \in X \setminus M$.*

Equivalently, a homogeneous relation H can be seen as a mapping from each $x \in X$ to a partition of $X \setminus \{x\}$, namely the equivalence classes of H_x. This generalizes graphs and *2-structures*, where modular decomposition still applies under the different but equivalent name of *clan decomposition* [11]. Roughly, a $2-$structure $G = (X, C)$ is a ground set X and an edge colouration $C : X^2 \to \mathbb{N}$ [11]. Thus, a digraph is a $2-$structure using two colours, denoting the existing (when $C(x, y) = 1$) and absent arcs (when $C(x, y) = 0$). There is no need of

the concept of *adjacency* nor *neighbourhood* nor *incidence* in a homogeneous relation! But a homogeneous relation is canonically derived from graphs and 2−structures as follows.

Definition 2 (Standard Homogeneous Relation). *[3,4] The* standard homogeneous relation $H(G)$ *of a 2-structure* $G = (X, C)$ *is*

$$H(G)(x|uv) \iff C(x,u) = C(x,v) \text{ and } C(u,x) = C(v,x).$$

Proposition 1. *Let G be a graph, or a tournament, or an oriented graph, or a directed graph, or a 2−structure. The modules of $H(G)$ exactly are the modules of G in the usual sense (see definitions in [15,21,11]).*

We now introduce the central notion of the paper which, from Proposition 3 (below), can be seen as a proper generalization of the classical modules/clans (c.f. [15,21,11]), and a dual notion to the generalized modules (c.f. [3,4]).

Definition 3 (Umodules). *A subset U of X is a* umodule *of H if*

$$\forall u, u' \in U, \quad \forall x, x' \in X \setminus U, \quad H(u|xx') \iff H(u'|xx').$$

Roughly, elements of a umodule come from the same "school of thinking": if one differentiates, resp. mixes together, some exterior elements, so does every element of the umodule (Fig. 1). A umodule U is *trivial* if $|U| \leq 1$ or if $|U| \geq |X| - 1$. The family of umodules of H is denoted by $\mathcal{U}_H$, and $\mathcal{U}$ when no confusion occurs. H is *umodular prime* if all its umodules are trivial. The following proposition links umodules to the 1-intersecting families framework as defined in [16]. The subsequent proposition tells how far umodules may generalize modules.

Proposition 2. *For any two umodules U, U' of a homogeneous relation H, if $U \cap U' \neq \emptyset$ then $U \cup U'$ is also a umodule of H.*

Proposition 3. *If H is a standard homogeneous relation (see Definition 2), then any module of H is a umodule of H. If H is an arbitrary homogeneous relation over a finite set X, then any module M of H is such that $X \setminus M$ is a umodule of H.*

In case of graphs, a natural question arises [9]: for which graphs the notions of module and umodule coincide? The following result, which can also be seen

Fig. 1. *left* Modules and umodules in a **graph**: $\{a, b\}$ is a module and also a umodule, $\{1, 2\}$ is a umodule but is not a module. *right* A **homogeneous relation** with a module which is not a umodule. $\{a, b\}$ is a module: they belong to the same equivalence class in both H_c and H_d. $\{a, b\}$ is not a umodule: c and d belong to the same class in H_a, and to different classes in H_b.

as a relaxed converse of Proposition 3, solves this problem. As with modules, let the umodules of a graph refer to those of its standard homogeneous relation. Notice here in a graph that the complementary of a umodule also is a umodule. A *threshold graph* is one that can be constructed from the single vertex by repeated additions of a single isolated or dominating vertex.

Proposition 4. *G is a threshold graph if and only if in all induced subgraph of G, every umodule is a module or the complementary of a module.*

Threshold graphs are known to be one of the smallest graph classes (see e.g. [2]). Therefore for most graphs umodules and modules differ, and Section 6 is devoted to the umodular graph decomposition. However, before deepening decomposition issues, let us first display umodule tractability.

3 Algorithmic Tractability for the General Case

As far as we are aware, there is no evidence of a decomposition scheme for arbitrary umodules. The first valuable objects to compute thus seem to be the maximal umodules with respect to some cut. Using this, we also provide a polynomial time algorithm computing the *strong* umodules (see definition afterwards).

3.1 Maximal Umodules with Respect to a Cut

Partitions will be ordered with respect to the usual partition lattice: $\mathcal{P} = \{P_1..P_p\}$ is *coarser* than $\mathcal{Q} = \{Q_1, \ldots, Q_q\}$, and $\mathcal{Q}$ is *thinner* than $\mathcal{P}$, if every part Q_i is contained in some P_j. It is noted $\mathcal{Q} \leq \mathcal{P}$ and $\mathcal{Q} < \mathcal{P}$ if the partitions are different. Let S be a subset of X. As the umodule family $\mathcal{U}$ is closed under union of intersecting members (Proposition 2), the inclusionwise maximal umodules included in either S or $X \setminus S$ form a partition of X, denoted by $MU(S) = MU(X \setminus S)$. In other words, this is the coarsest partition of X into umodules of H, which is thinner than $\{S, X \setminus S\}$. Roughly, it gives an indication on how the umodules are structured with respect to S: a umodule either is included in a umodule of $MU(S)$, or properly intersects S, or properly intersects $X \setminus S$, or trivial.

Definition 4. *For every subset $C \subseteq X$, the relation R_C on C is defined as:*
$$\forall x, y \in C, R_C(x, y) \text{ if } \forall a, b \in (X \setminus C) \quad H(x|ab) \Longleftrightarrow H(y|ab).$$

This clearly is an equivalence relation on C. Furthermore, C is a umodule if and only if R_C only has one equivalence class. Let us define a *refinement* operation, the main algorithmic tool for constructing $MU(S)$.

Definition 5. *Let $\mathcal{P}$ be a partition of X and C a part of $\mathcal{P}$. Let $C_1, \ldots, C_k$ be the equivalence classes of R_C. $Refine(\mathcal{P}, C)$ is the partition obtained from $\mathcal{P}$, by replacing part C by the parts $C_1, \ldots, C_k$. A partition $\mathcal{P}$ is* refinable *by C if $Refine(\mathcal{P}, C) \neq \mathcal{P}$. $\mathcal{P}$ is* unrefinable *if for every part C of $\mathcal{P}$, we have $\mathcal{P} = Refine(\mathcal{P}, C)$.*

$\mathcal{P} \leftarrow \{S, X \setminus S\}$
while *there exists an unmarked part* C *in* $\mathcal{P}$ **do**
  if $\mathcal{P} = Refine(\mathcal{P}, C)$ **then** mark C
  else $\mathcal{P} \leftarrow Refine(\mathcal{P}, C)$

Algorithm 1. Refinement algorithm computing $MU(S)$ given homogeneous relation H over X and $S \subseteq X$

Lemma 1. *Let H be a homogeneous relation over X, U a umodule of H, and $\mathcal{P}$ a partition of X. If U is included in a part of $\mathcal{P}$, then for any part C of $\mathcal{P}$, U is included in a part of $Refine(\mathcal{P}, C)$. Moreover, a part C of $\mathcal{P}$ is a umodule if and only if $\mathcal{P}$ is not refinable by C.*

Correctness of Algorithm 1 follows from Lemma 1 and the invariant: *There is no umodule partition $\mathcal{Q}$ such that $\mathcal{P} < \mathcal{Q} < \{S, X \setminus S\}$*. So, starting from $\{S, X \setminus S\}$ the algorithm constructs a strictly decreasing chain of partitions of X ending at $MU(S)$. Let us see how to implement it efficiently.

Lemma 2. *It is possible to compute $Refine(\mathcal{P}, C)$ in $O(|X|^2)$ time.*

Due to lack of space, the proof is omitted (see [5]). This lemma leads to an $O(|X|^3)$ time implementation of Algorithm 1. However:

Theorem 1. *For every $S \subseteq X$, $MU(S)$, the coarsest umodule partition thinner than $\{S, X - S\}$ can be computed in $O(|X|^2 \log |X|)$ time.*

Proof. We borrow on the well-known Hopcroft's partition refinement rule [24] and avoid at each step to consider the biggest part. Thus, to compute $MU(S)$ assuming that $|S| \leq |X - S|$, we first partition $X - A$ using the "neighbourhoods lists" of all $a \in A$. If we assume a data structure which links each edge ay to its opposite edge ya. We can associate in the meantime to each element $a \in A$ a bitvector representing how $X - A$ sees a. These $|A|$ bitvectors of size $|X - A|$ can be sorted in $O(|X|.|X - A|) \in O(|X|^2)$. Using Hopcroft's rule, a vertex a can only be explored at most $O(\log |X|)$ time, which yields the announced complexity. □

3.2 Strong Umodules Computation and Primality Test

A umodule is *strong* if it overlaps no other umodules, where two subsets overlap if none of the intersection and differences are empty. As two strong umodules are either disjoint, or one contains another, they can be ordered by inclusion into a tree (see e.g. laminar families in [25]). The following theorem answers both maximal umodule computation and primality test since a non-trivial umodule exists if and only if a non-trivial strong umodule exists.

Theorem 2. *There exists an $O(|X|^4 \log |X|)$ algorithm to compute the inclusion tree of strong umodules.*

Proof. Consider a non-trivial strong umodule M. For each pairwise distinct $x, y \notin M$ (at least two of them exist as M is not trivial), M is contained in

exactly one set of $MU(\{x, y\})$. The intersection of all these sets is exactly M. Indeed if it were M' with $M \subsetneq M'$ then there would exist $x \in M' \setminus M$. For $y \notin M$, $MU(\{x, y\})$ contains a umodule M'' smaller than M' but containing M, a contradiction. Then the algorithm could be: for any pair $\{x, y\}$ compute $MU(\{x, y\})$ in $O(|X|^2 \log |X|)$ time (Theorem 1), which is a family of at most $|X|^3$ umodules. Add the trivial umodules and greedily compute the intersection of overlapping umodules. It is possible in $O(|X|^4 \log |X|)$ time: for each triple (a, b, c) look for the umodules containing exactly two of them, they overlap. Then we have all strong umodules, and just have to order them into a tree. □

4 Two Decomposition Scenarios

Obviously, the number of umodules is potentially $2^{|X|}$. But we shall now focus on families having a polynomial-size representation. The umodules of local congruence 2 relations and self-complemented umodules families have such property: they can be stored in $O(|X|^2)$ and $O(|X|)$ space, respectively.

4.1 Local Congruence and Crossing Families

Definition 6 (Local congruence). *Let H be a homogeneous relation on X. For $x \in X$, the* congruence *of x is the maximal number of elements that x pairwise distinguishes (*i.e. *the number of equivalence classes of H_x). The* local congruence *of H is the maximum congruence of the elements of X.*

Standard homogeneous relations of undirected graphs and tournaments have local congruence 2. This value is 3 for antisymmetric digraphs and directed acyclic graphs, and is 4 for digraphs. When H has local congruence 2 (*LC2 condition* for short), we obtain the following structural property.

Definition 7 (Crossing family). *$\mathcal{F} \subseteq 2^X$ is a crossing family if, for any $A, B \in \mathcal{F}$, that $A \cap B \neq \emptyset$ and $A \cup B \neq X$ implies $A \cap B \in \mathcal{F}$ and $A \cup B \in \mathcal{F}$ (see e.g. [25] for further details).*

Proposition 5. *The umodules of a homogeneous relation with local congruence* 2 *form a crossing family, and can thus be stored in $O(|X|^2)$ space.*

Crossing families commonly arise as the minimizers of a submodular function, such as the family of minimum $s, t-$cuts of a network. For those families Gabow gave a compact representation in $O(|X|^2)$ space using a tree representation [14].

4.2 Self-complementarity and Bipartitive Families

A consequence of previous proposition is that standard homogeneous relations of graphs and tournaments have crossing umodules families. But they have stronger properties, used there to build a $O(|X|)$ space encoding of the umodules family.

Definition 8. *H fulfils the* four elements condition *if*
$\forall\ m, m', x, x' \in X, \begin{cases} H(m|xx') \wedge H(m'|xx') \wedge H(x|mm') \Rightarrow H(x'|mm') \\ \neg H(m|xx') \wedge \neg H(m'|xx') \wedge \neg H(x|mm') \Rightarrow \neg H(x'|mm') \end{cases}$.

Proposition 6. *Standard homogeneous relations of undirected graphs and tournaments satisfy the four elements condition.*

This is a light regularity condition, allowing to avoid examples similar to that of Fig. 1.*right*. Surprisingly enough, it suffices to make the umodule family behave in a very tractable manner (Proposition 7 and Corollary 1 below).

Definition 9 (Self-complementary condition). *A family $\mathcal{F}$ of subsets of X is* self-complemented *if for every subset A, $A \in \mathcal{F}$ implies $X \setminus A \in \mathcal{F}$.*

Proposition 7. *If a homogeneous relation H fulfils the four elements condition then the family $\mathcal{U}$ of umodules of H is self-complemented.*

The *four elements condition* allows to shrink a umodule, hence apply the divide and conquer paradigm to solve optimisation problems. However, as far as umodules are concerned, the *self-complementary* relaxation is sufficient to describe a tree-decomposition theorem as can be seen below. Finally, notice that the converse of Proposition 7 does not necessarily hold. The characterisation of relations having a self-complemented umodule family by a local axiom, such as the four elements condition, actually appears to be more difficult.

The following results on bipartitions can be found in [10] under the name of "decomposition frame with the intersection and transitivity properties", in [22] under the name of "bipartitive families" (the formalism used in this paper), and in [19] under the name of "unrooted set families". We call $\{X_i^1, X_i^2\}$ a *bipartition* of X if $X_i^1 \cup X_i^2 = X$ and $X_i^1 \cap X_i^2 = \emptyset$. Two bipartitions $\{X_i^1, X_i^2\}$ and $\{X_j^1, X_j^2\}$ *overlap* if for all $a, b = 1, 2$ the four intersections $X_i^a \cap X_j^b$ are not empty. A bipartition is *trivial* if one of the two parts is of size 1. Let $\mathcal{B} = \{\{X_i^1, X_i^2\}_{i \in 1,\ldots,k}\}$ be a family of k bipartitions of X. The *strong* bipartitions of $\mathcal{B}$ are those that do not overlap any other bipartition of $\mathcal{B}$. For instance, the trivial bipartitions of $\mathcal{B}$ are strong bipartitions of $\mathcal{B}$.

Proposition 8. *If $\mathcal{B}$ contains all trivial bipartitions of X, then there exists a unique tree $T(\mathcal{B})$*

- *with $|X|$ leaves, each leaf being labelled by an element of X.*
- *such that each edge e of $T(\mathcal{B})$ correspond to a strong bipartition of $\mathcal{B}$: the leaf labels of the two connected components of $T - e$ are exactly the two parts of a strong bipartition, and the converse also holds.*

Let N be a node of $T(\mathcal{B})$ of degree k. The labels of the leaves of the connected components of $T - N$ form a partition $X_1, \ldots, X_k$ of X. For $I \subseteq \{1, \ldots, k\}$ with $1 < |I| < k$, the bipartition $B(I)$ is $\{\cup_{i \in I} X_i, X \setminus \cup_{i \in I} X_i\}$.

Definition 10 (Bipartitive Family). *A family of bipartitions is a* bipartitive family *if it contains all the trivial bipartitions and if, for two overlapping bipartitions $\{X_i^1, X_i^2\}$ and $\{X_j^1, X_j^2\}$, the four bipartitions $\{X_i^a \cup X_j^b, X \setminus (X_i^a \cup X_j^b)\}$ (for all $a, b = 1, 2$) belong to $\mathcal{B}$.*

Theorem 3. *[10,22]If $\mathcal{B}$ is a bipartitive family, the nodes of $T(\mathcal{B})$ can be labelled* complete*,* circular *or* prime*, and the children of the* circular *nodes can be ordered in such a way that:*

- *If N is a complete node, for any $I \subseteq \{1,\ldots,k\}$ such that $1 < |I| < k$, $B(I) \in \mathcal{B}$.*
- *If N is a circular node, for any interval $I = [a,\ldots,b]$ of $\{1,\ldots,k\}$ such that $1 < |b-a| < k$, $B(I) \in \mathcal{B}$.*
- *If N is a prime node, for any element $I = \{a\}$ of $\{1,\ldots,k\}$ $B(I) \in \mathcal{B}$.*
- *There are no more bipartitions in $\mathcal{B}$ than the ones described above.*

For a bipartitive family $\mathcal{B}$, the labelled tree $T(\mathcal{B})$ is an $O(|X|)$-sized representation of $\mathcal{B}$, while the family can have up to $2^{|X|-1}-1$ bipartitions of $|X|$ elements each. This allows to efficiently perform algorithmic operations on $\mathcal{B}$. Notice that any self-complemented subset family can be seen as a family of bipartitions.

Proposition 9. *A self-complemented umodule family is bipartitive.*

Corollary 1 (Decomposition Theorem). *There is a unique $O(|X|)$-sized tree that gives a description of all possible umodules of a homogeneous relation H fulfilling the self-complementary condition. This tree is henceforth called* umodular decomposition tree*. Notice that it is an unrooted tree.*

Let H be a self-complemented homogeneous relation, $T(H)$ its umodular decomposition tree, and U a nontrivial strong umodule (if any). Let us examine some consequences of Theorem 3. Two umodules overlap if and only if they are incident to the same node of $T(H)$. As H is self-complemented the union of two overlapping umodules is a umodule (Proposition 2) but also their intersection. The strong umodule U is an edge in $T(H)$ incident with two nodes A and B.

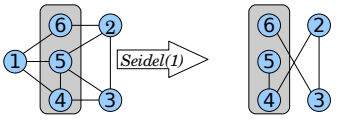
- If one of them, say A, is labelled prime then for any $x, y \notin U$ such that the least common ancestor of them in $T(H)$ is A, then $U \in MU(\{x,y\})$.
- If one of them, say A, is labelled circular then $U \in MU(\{x,y\})$ for any x belonging to the subtree rooted at the successor of U in the ordered circular list of A, and for any y belonging to the subtree rooted at its predecessor.
- If one of them, say A, is labelled complete then the intersection, for all $x, y \notin U$ whose least common ancestor is A, the intersection of all parts of $MU(\{x,y\})$ containing U is exactly U.

Theorem 2 then can be used to compute the strong umodule inclusion tree. After this, typing the nodes and ordering their sons according to the above definition is straightforward. Hence,

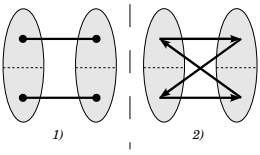
Theorem 4. *There exists an $O(|X|^4 \log(|X|))$ algorithm to compute the unique decomposition tree for a self complemented umodule family.*

5 Seidel-Switching Theorem: More Tractability

Standard homogeneous relations of graphs and tournaments satisfy both *LC2* and self-complemented conditions. We thus can decompose their umodules using



(a) An example of a Seidel switch on an undirected graph



(b) 1. A bi-join (i.e. umodule) in an undirected graph, 2. a umodule in a tournament

Fig. 2. (a) Seidel switching, (b) umodules on undirected graphs

either the crossing decomposition or the bipartitive decomposition. Moreover, relations that satisfy both conditions seem to own stronger potential. In particular, there is a local transformation from the umodules of such a relation to the modules of another relation. This operation was first introduced by J. Seidel in [26] on undirected graphs. It was later studied by several authors interested in some computational or structural aspects [8,17,18,20]. Following [18], we call it *Seidel switching*. We generalize it to homogeneous relations but take a restricted case of switch, with the slight difference that we remove from the transformation an element (see Fig. 2(a)). If H is a homogeneous relation on X and $s \in X$, we denote the equivalence classes of H_s by $H_s^1, \ldots, H_s^k$.

Definition 11 (Seidel switch). *Let H be a homogeneous relation of local congruence 2 on X, and s an element of X. The* Seidel switch *at s transforms H into the homogeneous relation $H(s)$ on $X \setminus \{s\}$ defined as*

$$\forall x \in X \setminus \{s\}, H(s)_x^1 = (H_x^1 \Delta H_s^j) \setminus \{s\} \text{ and } H(s)_x^2 = (H_x^2 \Delta H_s^j) \setminus \{s\}$$

with j such that $x \notin H_s^j$. $A \Delta B$ denotes the symmetric difference of A and B.

Theorem 5 (Seidel-switching Theorem). *Let H be a homogeneous relation of local congruence 2 on X such that $\mathfrak{U}_H$ is self-complemented. Let s be a member of X, and $U \subseteq X$ a subset containing s. Then, U is a umodule of H if and only if $M = X \setminus U$ is a module of the Seidel switch $H(s)$.*

Corollary 2. *The umodular decomposition tree of a self-complemented homogeneous relation of local congruence 2 on X can be computed in $O(|X|^2)$ time.*

Proof. A Seidel switch on any element will result in a relation where every module of the relation also is a umodule (c.f. *modular quotient* property [3]). Then, the $O(|X|^2)$-time algorithm depicted in [3] can be used. As two complemented strong umodules $M, X \setminus M$ of H, for $s \notin M$, correspond to a strong module M of $H(s)$, then the strong umodules of H can be found trivially from the strong modules of $H(s)$. Typing and ordering their sons is then easy. □

Notice that the modular decomposition tree of H can be trivial, while the one of its Seidel switch at s may be not. Besides, there is no real need to type and order

the sons of a node, as so-called *linear* nodes of the modular decomposition tree give circular nodes of the umodular decomposition tree with the same ordering of their sons, complete nodes of $H(s)$ give complete nodes of H and prime nodes of $H(s)$ give prime nodes of H. However, modular decomposition of homogeneous relations will not be discussed here, the reader should refer to [3].

6 Umodular Decomposition of Graphs and Tournaments

Let us now apply umodular decomposition to two well-known combinatorial objects: undirected graphs and tournaments. In this section we always implicitly refer to their standard homogeneous relations, for instance "the umodules of the graph G" stands for "the umodules of the standard homogeneous relation $H(G)$ of the graph G" and so on. Also, "graph" stands for "undirected graph". As we have seen, graphs and tournaments fulfil the four elements conditions, are of local congruence two, and their umodule family is self-complemented.

6.1 Bijoin Decomposition

Let us call *bijoin* a umodule of a graph or of a tournament. From definition, one can see what bijoins are (Fig.2(b)). In a graph, B is a bijoin if $X \setminus B$ can be partitioned in two sets C and D such that for each $x \in B$, either $N(x) \cap C = \emptyset$ and $D \subseteq N(x)$, or $N(x) \cap D = \emptyset$ and $C \subseteq N(x)$. For a tournament, same definition with $C \subseteq N^+(x)$ and $D \subseteq N^-(x)$, or $D \subseteq N^+(x)$ and $C \subseteq N^-(x)$.

Bijoins of graphs were studied in [23] as a new graph decomposition, generalizing modular decomposition. The Seidel switch was used to derive most of the properties claimed, especially a decomposition tree (with no *circular* nodes), a linear-time decomposition algorithm, a characterisation of the two kinds of *complete* nodes, and characterisation of *totally decomposable* graphs (see below). Bijoins of tournaments form a new decomposition. Their decomposition tree exists thanks to Corollary 1, since the bijoins form a self-complemented $LC2$ family. The tree computation follows from Corollary 2, that is

Proposition 10. *The umodular (bijoin) decomposition tree computation time of a tournament is* $O(|X|^2)$.

Proposition 11. *The umodular (bijoin) decomposition tree of a tournament has no* complete node. *And there exists a circular ordering of the vertices of the tournament such that every umodule of the tournament is a factor (interval) of this circular ordering.*

The first assumption can be checked by reader: it is impossible to build tournaments with more than four elements such that every vertex subset is a bijoin. The second is a consequence of the first, and of definitions in Theorem 3. As a

consequence, there are $O(|X|^2)$ bijoins in a tournament (the exponential growth of a bipartitive family comes from *complete* nodes).

6.2 Total Decomposability

Given a graph decomposition scheme, it is often worth to consider the totally decomposable graphs w.r.t. that scheme, namely those in which every "large enough" subgraph admits a non trivial decomposition. Generally this leads to the definition of very interesting graph classes, such as cographs with modular decomposition or distance hereditary graphs with split decomposition.

Theorem 6. *[23] The totally decomposable graphs w.r.t. bijoin decomposition are the (C_5,bull,gem,co-gem)-free graphs, and also exactly the graphs that can be obtained from a single vertex by a sequence of (twin,antitwin)-extensions.*

Definition 12. *The* in-diamond *(resp.* out-diamond*) is a tournament made with a cycle of three vertices plus a sink (resp. a source). A* diamond *is either an in- or an out-diamond. A tournament T is* locally transitive *if for each vertex $x \in V(T)$, both $T_{[N^+(x)]}$ and $T_{[N^-(x)]}$ are transitive. Two vertices x and y of a tournaments are* twins *if $N^+(x) \setminus \{y\} = N^+(y) \setminus \{x\}$ and* antitwins *if $N^+(x) \setminus \{y\} = N^-(y) \setminus \{x\}$. An* extension *of x by a twin (resp. antitwin) y consists in adding a new vertex y to T and making y twin (resp. antitwin) of x.*

Theorem 7. *Let T be a tournament. The following propositions are equivalent:*

1. *T is diamond-free (no induced subgraph is a diamond)*
2. *T is locally transitive*
3. *T is totally decomposable with respect to bijoin decomposition*
4. *T is obtained from a single vertex by a sequence of (twin,antitwin)-extensions.*

Due to lack of space, the proof is omitted (see [5]). As the umodular decomposition tree of a totally decomposable tournament may have no *prime* nodes, and since two *circular* nodes may not be adjacent, it is not hard to check that the umodular decomposition tree of a totally decomposable tournament has only a single *circular* node. The ordering of the vertices along this node is known as *circular ordering*. This ordering is such that, for each vertex x, the vertices of $N^+(x)$ follow consecutively; and so do vertices from $N^-(x)$. This, combined to the above theorem, could be seen as a sketched proof of the characterisation of *round tournaments* by local transitivity (see e.g. [1] for further information).

In the extended version [5], we present an $O(n^2)$ recognition algorithm, making an intensive use of this ordering property, and computing this ordering. It allows us to solve the isomorphism problem for the class of such tournament in $O(n^2)$ time, like in [7]. We also propose the first linear-time algorithm for the feedback vertex set problems (NP-complete for tournaments). The basic idea is to find a vertex of highest outgoing degree, and output the tournament composed of this vertex and its outgoing neighbourhood.

References

1. Bang-Jensen, J., Gutin, G.: Digraphs: Theory, Algorithms and Applications. Springer Monographs in Mathematics. Springer, Heidelberg (2001)
2. Brandstadt, A., Le, V.B., Spinrad, J.P.: Graph Classes: A Survey. SIAM Monographs on Discrete Mathematics and Applications. SIAM (1999)
3. Bui-Xuan, B.-M., Habib, M., Limouzy, V., de Montgolfier, F.: Algorithmic aspects of a general modular decomposition theory. Technical report (to appear)
4. Bui Xuan, B.-M., Habib, M., Limouzy, V., de Montgolfier, F.: Homogeneity vs. adjacency: generalising some graph decomposition algorithms. In: Fomin, F.V. (ed.) WG 2006. LNCS, vol. 4271, Springer, Heidelberg (2006)
5. Bui-Xuan, B.-M., Habib, M., Limouzy, V., de Montgolfier, F.: A new tractable decomposition. Technical report (2007), http://hal-lirmm.ccsd.cnrs.fr/lirmm-00157502
6. Chein, M., Habib, M., Maurer, M.C.: Partitive hypergraphs. Discrete Mathematics 37(1), 35–50 (1981)
7. Clarou, E.: Une hiérarchie de forçage pour les tournois indécomposables. PhD thesis, Université Claude Bernard Lyon I (1996)
8. Colbourn, C.J., Corneil, D.G.: On deciding switching equivalence of graphs. Discrete Applied Mathematics 2(3), 181–184 (1980)
9. Corneil, D.G.: Private communication. Dagstuhl (2007)
10. Cunningham, W.H.: A combinatorial decomposition theory. PhD thesis, University of Waterloo, Waterloo, Ontario, Canada (1973)
11. Ehrenfeucht, A., Harju, T., Rozenberg, G.: The Theory of 2-Structures- A Framework for Decomposition and Transformation of Graphs. World Scientific, Singapore (1999)
12. Everett, M.G., Borgatti, S.P.: Role colouring a graph. Mathematical Social Sciences 21, 183–188 (1991)
13. Fiala, J., Paulusma, D.: The computational complexity of the role assignment problem. In: Baeten, J.C.M., Lenstra, J.K., Parrow, J., Woeginger, G.J. (eds.) ICALP 2003. LNCS, vol. 2719, pp. 817–828. Springer, Heidelberg (2003)
14. Gabow, H.N.: Centroids, representations, and submodular flows. Journal of Algorithms 18(3), 586–628 (1995)
15. Gallai, T.: Transitiv orientierbare Graphen. Acta Math. Acad. Sci. Hungar. 18, 25–66 (1967)
16. Habib, M., Maurer, M.C.: 1-intersecting families. Discrete Mathematics 53, 91–101 (1985)
17. Hayward, R.B.: Recognizing 3-structure: A switching approach. Journal of Combinatorial Theory, Series B 66(2), 247–262 (1996)
18. Hertz, A.: On perfect switching classes. Discrete Applied Mathematics 94(1-3), 3–7 (1999)
19. Hsu, W.-L., McConnell, R.M.: PC-trees and circular-ones arrangements. Theoretical Computer Science 296, 99–116 (2003)
20. Kratochvíl, J., Nešetřil, J., Zýka, O.: On the computational complexity of Seidel's switching. In: Kratochvíl, J., Nešetřil, J. (eds.) Fourth Czechoslovakian Symposium on Combinatorics, Graphs and Complexity (Prachatice, 1990). Ann. Discrete Math, vol. 51, pp. 161–166. North-Holland, Amsterdam (1992)
21. Möhring, R.H., Radermacher, F.J.: Substitution decomposition for discrete structures and connections with combinatorial optimization. Annals of Discrete Mathematics 19, 257–356 (1984)

22. de Montgolfier, F.: Décomposition modulaire des graphes. Théorie, extensions et algorithmes. PhD thesis, Université Montpellier II (2003)
23. de Montgolfier, F., Rao, M.: The bi-join decomposition. In: ICGT 2005, 7th International Colloquium on Graph Theory (2005)
24. Paige, R., Tarjan, R.E.: Three partition refinement algorithms. SIAM Journal on Computing 16(6), 973–989 (1987)
25. Schrijver, A.: Combinatorial Optimization - Polyhedra and Efficiency. Springer, Heidelberg (2003)
26. Seidel, J.J.: A survey of two-graphs. In: Colloquio Internazionale sulle Teorie Combinatorie (Rome, 1973), Tomo I, pp. 481–511. Atti dei Convegni Lincei, No. 17. Accad. Naz. Lincei, Rome (1976)
27. White, D.R., Reitz, K.P.: Graph and semigroup homomorphisms on networks of relations. Social Networks 5, 193–234 (1983)

Escaping Off-Line Searchers and a Discrete Isoperimetric Theorem

Peter Brass[1], Kyue D. Kim[1], Hyeon-Suk Na[2,⋆], and Chan-Su Shin[3,⋆⋆]

[1] Department of Computer Science, City College, New York, USA
peter@cs.ccny.cuny.edu, quoderatd@gmail.com
[2] School of Computing, Soongsil University, Seoul, Korea
hsnaa@ssu.ac.kr
[3] School of Electrical and Information Engineering, Hankuk University of Foreign Studies, Yongin, Korea
cssin@hufs.ac.kr

Abstract. Given a set of searchers in the grid, whose search paths are known in advance, can a target that moves at the same speed as the searchers escape detection indefinitely? We study the number of searchers against which the target can still escape. This is less than n in an $n \times n$ grid, since a row of searchers can sweep the allowed region.

In an alternating move model where at each time first all searchers move and then the target moves, we show that a target can always escape $\lfloor \frac{1}{2}n \rfloor$ searchers and there is a strategy for $\lfloor \frac{1}{2}n \rfloor + 1$ searchers to catch the target. This improves a recent bound $\Omega(\sqrt{n})$ [5] in the simultaneous move model. We also prove similar bounds for the continuous analogue, as well as for searchers and targets moving with different speeds. In the proof, we use a new isoperimetric theorem for subsets of the $n \times n$ grid, which is of independent interest.

1 Introduction

Pursuit-Evasion problems have been studied in many models, and under many names, like lion and man [14] or hunter and rabbit [1]. In each case, there is a target t and one or more searchers $s_1, \ldots, s_k$; both target and searchers move, and the searchers aim to catch the target. The problems differ by the domain of the movement, which might be a graph [8,12,13], the entire plane [14], or some bounded domain [7], by relative speed of target and searchers, and by the information that the searchers receive about the target: they might know its position [14], or be constrained by visibility [6,9,11,15], or recognize the target only if in their detection range [7], in the graph case, if they occupy the same node.

In a recent paper, Dumitrescu et al. [5] introduced the model of *offline* searchers. Here, the search path for each searcher is known in advance, for all future, and the question is whether the target t can escape these searchers for

⋆ Corresponding author. Supported by Soongsil University Research Fund.
⋆⋆ Supported by Research Grant 2007, Hankuk University of Foreign Studies.

T. Tokuyama (Ed.): ISAAC 2007, LNCS 4835, pp. 65–74, 2007.

an arbitrary long time, using the information of those given search paths. If the domain is an $n \times n$ grid, we can arrange a row of n searchers that sweeps this grid square, and the target cannot escape even if it knows those search paths in advance. But if the number of searchers is smaller, the target may escape detection. Dumitrescu et al. [5] proved that the target can always escape $O(\sqrt{n})$ offline searchers in the $n \times n$ grid. We improve this bound.

Theorem 1. *In an $n \times n$ grid, a target can always escape $\lfloor \frac{n}{2} \rfloor$ offline searchers. In an alternating move model where at each time first all searchers move and then the target moves, there is a strategy for $\lfloor \frac{n}{2} \rfloor + 1$ searchers to catch the target.*

To prove the first part of the theorem, in Section 2.1, we use the following discrete isoperimetric theorem for finite grid graphs, which will be proved in Section 3.

Theorem 2. *If X is a subset of the vertices of the $n \times n$ grid, and $\mathrm{bd}(X)$ is the set of points in X that have a grid-neighbor not in X, then*

- *If $\frac{1}{2}(i-1)i + 1 \le |X| \le n^2 - \frac{1}{2}(i-2)(i-1) - 1$ for some $1 \le i \le n$, then $|\mathrm{bd}(X)| \ge i$.*
- *If $|X| = n^2$, then $|\mathrm{bd}(X)| = 0$.*

All these bounds are best possible.

This is a discrete isoperimetric theorem for the grid graph; similar theorems for the grid, unbounded or wrapped to a torus, have been studied in a number of papers [2,3,4,10]; but in this result, the boundary effects are important.

In Section 4, we use the same proof technique used in Theorem 1 to obtain the results in different situations; searchers and target with different speeds or in the continuous space and time. The following theorems will be proved in Section 4. Theorem 4 improves the recent bound $\Omega(\sqrt{n})$ on the number of searchers such that a target can escape in the continuous model [5].

Theorem 3. *Suppose that the searchers are v times as fast as the target in an $n \times n$ grid for some integer $v \ge 1$. Then the target can always escape $\lfloor \frac{n}{v+1} \rfloor$ offline searchers, and there is a strategy for $\lceil \frac{n}{v+1} \rceil + 1$ searchers to catch the target in the alternating move model.*

Theorem 4. *A target can always escape cn offline searchers in a square with side length n for some $c > 0$.*

2 Escaping Offline Searchers in the Grid Square

An $n \times n$ grid $G_n = (V, E)$ $(n \ge 2)$ has n^2 vertices with integer coordinates $[1, n] \times [1, n]$. In the following, we always assume that there are k searchers $s_1, \ldots, s_k$ and one target t. Their positions are the vertices of G_n, and any move either goes to a neighboring vertex, or stays in the same vertex. So if a current position is vertex v, then the possible next positions are the vertices of the closed neighborhood $N(v)$ (including v). The searchers and the target

move alternatingly, at time a, first all searchers move, with s_i moving from $s_i(a-1)$ to $s_i(a)$, then the target t moves from $t(a-1)$ to $t(a)$. The target *escapes detection* if for all times a and all searchers s_i it holds that $t(a) \neq s_i(a)$ and $t(a) \neq s_i(a+1)$. The searcher paths are given, and we want to choose a target path that escapes detection. Note that if a target can escape k searchers in our "alternating move" model, then this target can also escape k searchers in the "simultaneous move" model of Dumitrescu et al. [5] and thus the first part of Theorem 1 and Theorems 3 and 4 improve the bound $\Omega(\sqrt{n})$ proven in [5]. However, the converse is not true; the strategy of searchers to catch the target in our "alternating move" model does not provide one in the "simultaneous move" model. In the following subsections, we prove Theorem 1: $\lfloor \frac{n}{2} \rfloor$ offline searchers are never enough to catch the target, and there is a strategy for $\lfloor \frac{n}{2} \rfloor + 1$ searchers to catch the target in the alternating move model, so this is tight.

2.1 $\lfloor \frac{n}{2} \rfloor$ Searchers Are Never Enough

We define $Forb(a, b)$ as the set of *forbidden* vertices $p \in V$ for which any target t with $t(a) = p$ will be caught by one of the searchers by time b at latest. This set satisfies a dynamic-programming like recursion: a vertex p at time a will be unavoidably captured if either it will be captured immediately at time a, or any vertex that could be reached by time $a+1$ will lead to unavoidable capture. This can be summarized in two formulas

$$Forb(a, a+1) = \bigcup_{i=1}^{k} \{s_i(a), s_i(a+1)\} \tag{1}$$

and

$$Forb(a, b) = Forb(a, a+1) \cup \{p \in V \mid N(p) \subset Forb(a+1, b)\}. \tag{2}$$

For fixed a and $b \to \infty$, the sets $Forb(a, b)$ form an increasing family of sets. The target t can avoid the searchers for arbitrary long, if the sets $Forb(a, b)$ never become the set of all vertices, i.e., $Forb(a, b) \neq V$. Let $forb(a, b) = |\, Forb(a, b)|$ be the number of forbidden points, then we have to show that for $k = \frac{1}{2}n$ and for fixed a and arbitrary large b, it holds that $forb(a, b) < n^2$. Indeed we will show a stronger statement that $forb(a, b) \leq \frac{1}{2}n^2$.

Since $|\, Forb(a, a+1)| \leq 2k$ and $\{p \in V \mid N(p) \subset Forb(a+1, b)\} \subset Forb(a+1, b)$, the above recursion (2) yields that $forb(a, b) \leq 2k + forb(a+1, b)$. This is actually overestimate of $forb(a, b)$, since in $\{p \mid N(p) \subset Forb(a+1, b)\}$ we lose all those points from $Forb(a+1, b)$ that have a neighbor not in $Forb(a+1, b)$. Let $\mathrm{bd}(X)$ denote for any $X \subset V$ the set of vertices of X which have a neighbor in $V \setminus X$. Then we have

$$forb(a, b) \leq 2k + forb(a+1, b) - |\, \mathrm{bd}(Forb(a+1, b))|. \tag{3}$$

Now $|\, \mathrm{bd}(Forb(a+1, b))|$ can be small if $Forb(a+1, b)$ is small or very large, but our key observation is that for a mid-sized set $Forb(a+1, b)$, its boundary size cannot be small, which is exactly what Theorem 2 shows.

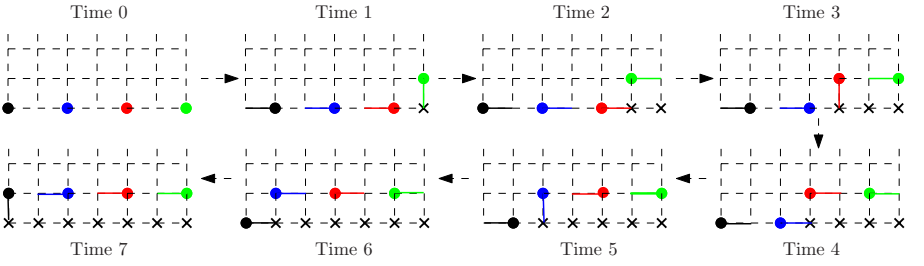
Using Theorem 2, we can now argue as follows. Fix $k = \lfloor \frac{n}{2} \rfloor$. If $forb(1, b)$ becomes close to n^2 when b becomes large, then there must be some b with $forb(1, b) > \frac{1}{2}n^2$. Consider now the sequence $forb(b-1, b)$, $forb(b-2, b)$,..., $forb(2, b)$, $forb(1, b)$. This sequence satisfies that (i) $forb(b-1, b) \leq 2k \leq n$, (ii) $forb(1, b) > \frac{1}{2}n^2$ and (iii) $forb(a, b) \leq forb(a+1, b) + n - |\operatorname{bd}(Forb(a+1, b))|$ for $1 \leq a \leq b-2$. Property (iii) implies that along the sequence each term grows by at most n, but Theorem 2 implies that the sequence grows slower when $forb(a+1, b)$ becomes near $\frac{n^2}{2}$. Indeed, if $\frac{1}{2}(\lceil \frac{n}{2} \rceil - 1)\lceil \frac{n}{2} \rceil + 1 \leq forb(a+1, b) \leq \frac{1}{2}n^2$, Theorem 2 shows that the boundary of $Forb(a+1, b)$ contains at least $\lceil \frac{n}{2} \rceil$ points and we have that $forb(a, b) \leq forb(a+1, b) + \lfloor \frac{n}{2} \rfloor$.

By (i) and (ii), there must be some a^* with $forb(a^*, b) \leq \frac{1}{2}n^2 < forb(a^* - 1, b)$. Then we have that $forb(a^*, b) > \frac{1}{2}n^2 - \lfloor \frac{n}{2} \rfloor$ (since otherwise $forb(a^* - 1, b)$ would be at most $\frac{1}{2}n^2$) and thus that $\frac{1}{2}n^2 - \frac{n}{2} < forb(a^*, b) \leq \frac{1}{2}n^2$. However, Theorem 2 again yields that the set $Forb(a^*, b)$ of this size must have at least n boundary points, and plugging this into inequality (3) gives that $forb(a^* - 1, b) \leq forb(a^*, b) + n - n \leq \frac{1}{2}n^2$, which contradicts to the definition of a^* that $forb(a^*, b) \leq \frac{1}{2}n^2 < forb(a^* - 1, b)$. This completes the proof of Theorem 1.

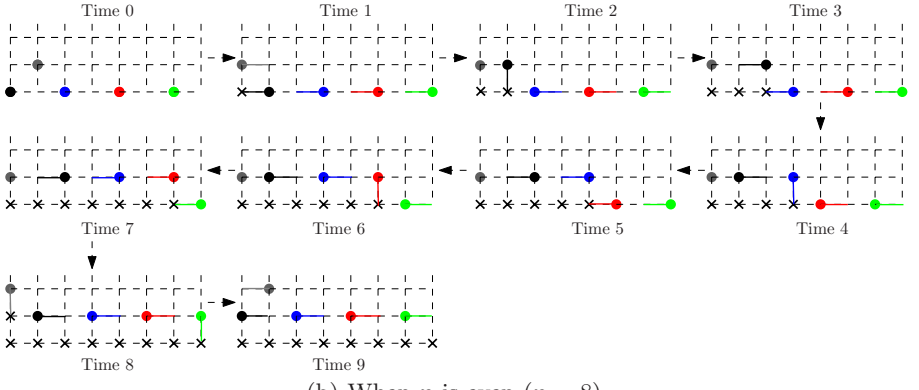
2.2 $\lfloor \frac{n}{2} \rfloor + 1$ Searchers Are Enough

We present a strategy for $\lfloor \frac{n}{2} \rfloor + 1$ searchers to catch the target in the alternating move model. As illustrated in Figure 1(a), the idea of the searchers' strategy that will catch any target is that a searcher moving back and forth between two consecutive grids blocks both positions; so a row in which searchers and gaps alternate, but the searchers move into and out of these gaps, cannot be passed by the target. We need one additional searcher to allow the searchers to move one row up, and finally to sweep the whole grid from the bottommost row to the topmost row.

Let $\hat{F}(a)$ be the set of $p \in V$ that the target cannot reach at time a without having been detected by that time. Refer to Fig. 1(a) for the illustration of $\hat{F}(a-1)$ at time a. The cross-mark at time 1 is the place where the green searcher was at time 0, so the target cannot lie there at time 0 and is contained in $\hat{F}(0)$. Two cross-marks at time 2 are forbidden at time 1 because of the red and green searchers, and are contained in $\hat{F}(1)$. If we can move the searchers at every time $a \geq 1$ so that every point in $N(\hat{F}(a-1))$ is either contained in $\hat{F}(a-1)$ or detected by some searchers at times $a-1$ or a, then those points are still unreachable by the target at time a; formally, if $N(\hat{F}(a-1)) \subseteq Forb(a-1, a) \cup \hat{F}(a-1)$, then $\hat{F}(a-1) \subseteq \hat{F}(a)$. Indeed, if $p \in \hat{F}(a-1)$ and $p = t(a)$, then $t(a-1) \in N(p) \subseteq Forb(a-1, a) \cup \hat{F}(a-1)$, which implies that $t(a-1)$ was unreachable by time $a-1$ or is caught by the searchers at times $a-1$ or a, and that p cannot be reached by the target without having been detected by time a. So our strategy is to extend $\hat{F}(a)$ row by row, besieging the target to the upper rows, and finally to leave no place for the target outside $\hat{F}(a)$. We need to handle the cases that n is even and n is odd, in a different manner. Fig. 1 illustrates the cases when $n = 7$ and $n = 8$. At time a, the cross-marks represent some vertices in $\hat{F}(a-1)$ and we can check that $N(\hat{F}(a-1)) \subseteq Forb(a-1, a) \cup \hat{F}(a-1)$.



(a) When n is odd ($n = 7$).



(b) When n is even ($n = 8$).

Fig. 1. Strategy for $\lfloor \frac{n}{2} \rfloor + 1$ searchers to catch the target; we only illustrate the procedure for the searchers to move one row up because this can be repeated to sweep all rows of the grid. At time a, solid disks with different colors represent searchers just moved from the other ends of the line segments, and the cross-marks represent some vertices in $\hat{F}(a-1)$.

3 Proof of the Isoperimetric Theorem

We now show that $|\operatorname{bd}(X)| \geq i$ for any $X \subseteq V$ with $\frac{1}{2}(i-1)i + 1 \leq |X| \leq n^2 - \frac{1}{2}(i-1)(i-2) - 1$. To prove this, we first convert X by the following operations to a staircase point-set X'' having the same number of points but no more boundary points than X, and then show that $|\operatorname{bd}(X'')| \geq i$.

Compression. Denote by $X_j \subseteq X$ the points of X in column j of G_n. First, compress the points of X to the bottom of G_n as much as possible, so that in each column the lowest point lies on the bottom row and the highest point lies at the $|X_j|$-th row. Then shift all the columns to the left so that the leftmost column becomes the first column of G_n without empty columns in between. Let us denote by X'_j the compressed column of X_j, and by X' the set of the compressed columns

of X. Since $|X_j| = |X'_j|$, it is clear that $|X'| = |X|$, but the number of boundary points may decrease. Indeed, we have that either $|\operatorname{bd}(X) \cap X_j| = |\operatorname{bd}(X') \cap X'_j| = 0$, or

$$|\operatorname{bd}(X) \cap X_j| \geq |\operatorname{bd}(X') \cap X'_j| = \max\{|X_j| - |X_{j-1}|, |X_j| - |X_{j+1}|, 1\}.$$

Thus we have that $|\operatorname{bd}(X)| \geq |\operatorname{bd}(X')|$.

Sorting. Now we sort the columns X'_j of X' according to their size (or height) $|X'_j|$ in non-increasing order from left to right. The resulting point set X'' is a staircase such that each column X''_j in X'' has the same height as its corresponding column $X'_{\sigma(j)}$ in X' for a permutation σ induced by sorting. Since the columns are sorted by their height, it holds for any column X''_j that $|X''_j| - |X''_{j+1}| \leq \max\{|X'_{\sigma(j)}| - |X'_{\sigma(j)+1}|, |X'_{\sigma(j)}| - |X'_{\sigma(j)-1}|, 1\}$. Thus $|\operatorname{bd}(X'') \cap X''_j| \leq |\operatorname{bd}(X') \cap X'_{\sigma(j)}|$, yielding that $|\operatorname{bd}(X'')| \leq |\operatorname{bd}(X')|$.

Proving that $|\operatorname{bd}(X'')| \geq i$. See Fig. 2 for the following definitions. Let weakbd(X'') be the set of points $(x, y) \in X''$ such that at least one of points $\{(x + i, y + j) : i, j = -1, 0, 1\}$ are in $V \setminus X''$. Let $C_{\max} := \max\{x + y : (x, y) \in \text{weakbd}(X'')\}$ and $C_{\min} := \min\{x+y : (x, y) \in \text{weakbd}(X'')\}$. Denote the points of X'' defining $C_{\max}$ and $C_{\min}$ by $p_{\max} = (x_{\max}, y_{\max})$ and $p_{\min} = (x_{\min}, y_{\min})$, respectively. Note that $\operatorname{bd}(X'') \subseteq \text{weakbd}(X'')$, $p_{\max} \in \operatorname{bd}(X'')$ and $p_{\min} \in \text{weakbd}(X'')$. No points of X'' lie above $C_{\max}$ and all points below $C_{\min}$ are in X''. The line $x + y = i$ contains at most $i - 1$ points in X'' for $i \leq n$ and at most $2n - i + 1$ points in X'' for $i > n$; refer to the line $x + y = 6$ in Fig. 2(a).

We now have a simple fact: as illustrated in Fig. 2(b), for any X'' with $\frac{1}{2}(i - 1)i + 1 \leq |X''| = |X| \leq n^2 - \frac{1}{2}(i - 1)(i - 2) - 1$, we have that $C_{\max} \geq i + 1$ and $C_{\min} \leq 2n - i$. Proving that $C_{\max} \geq i + 1$ is immediate from the observation that the number of points in X'' with $C_{\max} \leq i$ cannot be larger than $\frac{1}{2}(i - 1)i$. Similarly, we can prove that $C_{\min} \leq 2n - i$.

Consider the case that $i + 1 \leq C_{\max} \leq n$. As shown in Fig. 3(a), the number of boundary points of X'' in the left-side of $p_{\max}$ is at least $x_{\max} - 1$ and the number of boundary points of X'' in the right-side of $p_{\max}$ is at least $y_{\max} - 1$, so counting $p_{\max}$ itself we have

$$|\operatorname{bd}(X'')| \geq x_{\max} + y_{\max} - 1 = C_{\max} - 1 \geq i.$$

Similarly, if $n \leq C_{\min} \leq 2n - i$, then the number of boundary points of X'' in the left-side of $p_{\min}$ is at least $n - y_{\min}$ and the number of boundary points of X'' in the right-side of $p_{\min}$ is at least $n - x_{\min}$. The point $p_{\min}$ might not be a boundary point of X'', thus we get

$$|\operatorname{bd}(X'')| \geq 2n - y_{\min} - x_{\min} = 2n - C_{\min} \geq i.$$

Now we have only one case left, that is, $C_{\max} > n$ and $C_{\min} < n$. We claim that in this case $|\operatorname{bd}(X'')| \geq n$, proving that $|\operatorname{bd}(X'')| \geq i$ for all i. For illustration, refer to Fig. 3(b). Consider the top leftmost point p_ℓ and the bottom rightmost

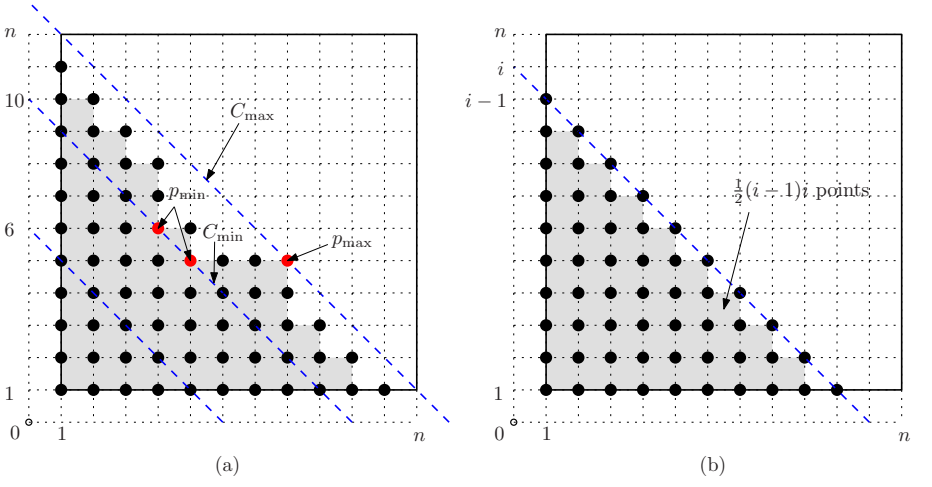


Fig. 2. (a) The definition of $C_{\max}$, $C_{\min}$, $p_{\max}$, and $p_{\min}$. Here $C_{\max} := \{(x, y) \mid x+y = n+1\}$ and $C_{\min} := \{(x, y) \mid x + y = 10\}$. (b) Since the line $x + y = i$ contains at most $i - 1$ points, for any X'' such that $|X''| \geq \frac{1}{2}(i-1)i+1$, it holds that $C_{\max} \geq i + 1$.

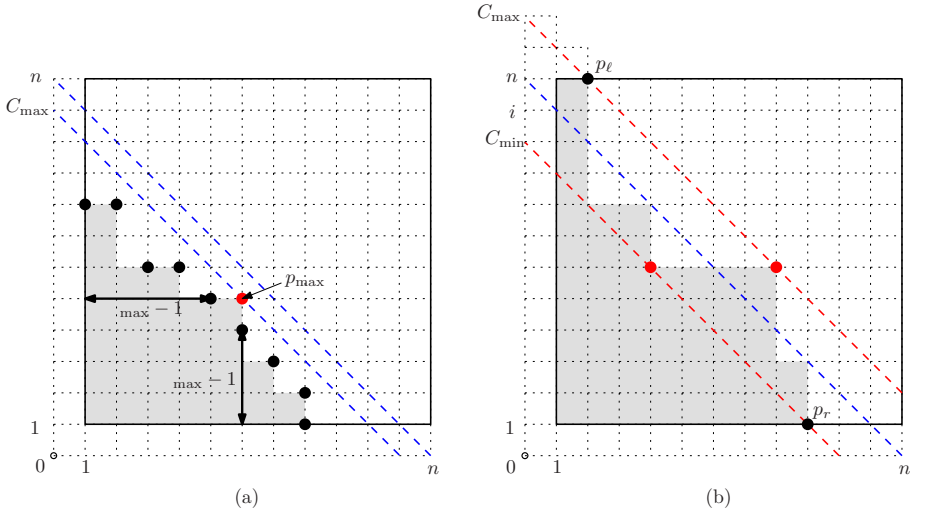


Fig. 3. (a) When $C_{\max} \leq n$, the number of boundary points of X'' is at least $x_{\max} + y_{\max} - 1 = C_{\max} - 1$, so $|\operatorname{bd}(X'')| \geq i$. (b) When p_ℓ lies above the line $x + y = n$ and p_r lies below the line, p_ℓ is on the topmost row of G_n and p_r is on the bottommost row of G_n. Thus the number of boundary points of X'' is at least n.

point p_r of $\operatorname{bd}(X'')$. If one of them, say p_ℓ, lies above the line $x + y = n$ and the other below line $x + y = n$, then p_ℓ must be on the topmost row of G_n and p_r must be on the bottommost row of G_n, which means, as shown in Fig. 3(b)

that the boundary of X'' contains at least n points. For the other case when p_ℓ lies below the line and p_r lies above the line, they are on the leftmost and rightmost row of G_n, so we have also that $|\operatorname{bd}(X'')| \geq n$. If both of p_ℓ and p_r lie below the line $x + y = n$, then from the assumption that $C_{\max} > n$, the number of boundary points of X'' in the left-side of $p_{\max}$ is at least $x_{\max} - 1$ and the number of boundary points of X'' in the right-side of $p_{\max}$ is at least $y_{\max} - 1$, which proves that $|\operatorname{bd}(X'')| \geq C_{\max} - 1 \geq n$. Similarly, if both of p_ℓ and p_r lie above line $x + y = n$, then from the assumption that $C_{\min} < n$, the number of boundary points of X'' in the left-side of $p_{\min}$ is at least $n - y_{\min}$ and the number of boundary points of X'' in the right-side of $p_{\min}$ is at least $n - x_{\min}$, proving that $|\operatorname{bd}(X'')| \geq 2n - C_{\min} \geq n$. Thus the proof of the isoperimetric theorem is completed.

4 Evading Offline Searchers in Related Models

Searchers and target of different speed. If the target is faster than the searchers, then the lower bound does not change; if $\lfloor \frac{1}{2}n \rfloor$ searchers are insufficient to catch the target of speed one, then they are also insufficient to catch a faster target. If the searchers are v times faster than the target, then the argument for the lower bound stays almost the same, but only we have $\mathit{forb}(a, a+1) \leq (v+1)k$, which shows that $\lfloor \frac{n}{v+1} \rfloor$ searchers are not sufficient to catch the target. We can catch the target with $\lceil \frac{n}{v+1} \rceil + 1$ searchers by simulating the strategy for $v = 1$ (Section 2.2) as follows; assign a searcher at every $(v+1)$ consecutive grid points on the bottommost row as in Figure 4, so that each searcher patrols the assigned $(v+1)$ grid points. Then $\lceil \frac{n}{v+1} \rceil$ searchers are set on the bottommost row. We assign one more searcher at the second bottommost row as in Figure 4. Then we can move the $\lceil \frac{n}{v+1} \rceil + 1$ searchers by similarly simulating the strategy used when n is even and $v = 1$. We here omit the details. As a result, a target of speed one can escape $\lfloor \frac{n}{v+1} \rfloor$ searchers of speed $v > 1$, but it cannot escape $\lceil \frac{n}{v+1} \rceil + 1$ searchers. Unlike $v = 1$, this is not tight because we need one more searcher when n is not multiple of $(v+1)$. Filling this gap remains open.

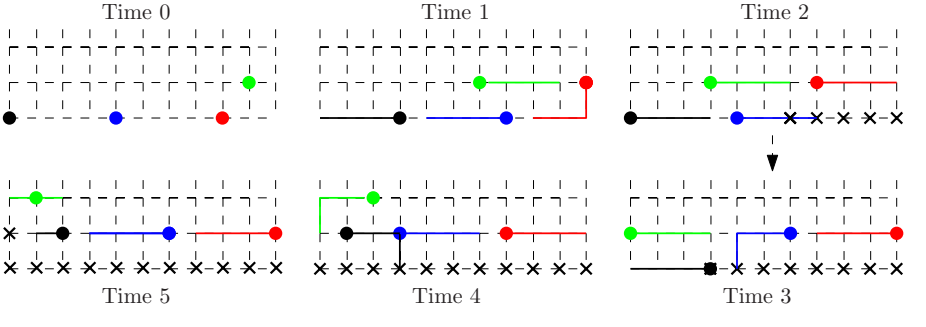


Fig. 4. When $v = 3$ and $n = 11$

Searchers and target in the continuous model. We have k searchers and one target moving with unit speed in a $2n \times 2n$ square S, and the target is detected if it comes within unit distance to a searcher. Again, n searchers are sufficient to create a row of searchers, which will always discover the target. We claim that for some $c > 0$, the target can always escape cn searchers. This improves the corresponding $\Omega(\sqrt{n})$ bound in [5].

The proof is by reduction to a similar problem as before. First we discretize the time. Instead of looking all the time at a disc of radius 1 around each searcher, we check for the moments $1, 2, 3, \ldots$ a disc of radius 3 around each searcher. We claim that this is a stronger searching: if t escapes these time-discrete observations, then t also escapes the original searchers; for if $d(t(x), s_i(x)) \leq 1$ for some $x \in [0, 1]$ and i, then $d(t(0), s_i(0)) \leq d(t(0), t(x)) + d(t(x), s_i(x)) + d(s_i(x), s_i(0)) \leq 3$. So we have to show that for some $c > 0$, the target can always escape $k = cn$ searchers, where the target and the searchers move in each step at most distance 1, and the target is detected if it comes within distance 3 of a searcher. But this problem again fits the previous proof; if $Forb(a, b)$ are those points of p in the square S, from which any target with $t(a) = p$ will unavoidably be caught by time b, and $B(p, r)$ is a disk centered at a point p with radius r, then we have almost the same relations:

$$Forb(a, a+1) = \bigcup_{i=1}^{k} B(s_i(a), 3) \cup B(s_i(a+1), 3),$$

and

$$Forb(a, b) = Forb(a, a+1) \cup \{p \in S \mid B(p, 1) \subset Forb(a+1, b)\}.$$

With $forb(a, b) = \text{area}(Forb(a, b))$, this gives $forb(a, a+1) \leq 9\pi k + 6$ and $forb(a, b) \leq forb(a+1, b) + 9\pi k + 6 - R$, where R is the total area of all points in $Forb(a+1, b)$ that are within distance at most one to points of the square that are outside $Forb(a+1, b)$. That area is $\Omega(n)$ if $Forb(a+1, b)$ contains a positive fraction of the entire area of the square; from this follows the claim: there is a $c > 0$ such that t can always escape cn searchers. This also holds even for the the searchers and target with different constant speeds by the similar argument in the discrete model.

5 Conclusion

We showed that there is some constant $0 < c < 1$ such that a target can always escape cn searchers in $n \times n$ square as well as in $n \times n$ grid even with different constant speeds in an alternating move model. These results drastically improve a recent result of $\Omega(\sqrt{n})$ [5], and the bounds are almost tight; for the searchers and target with the same speed, the bound is tight.

An interesting open question is to find the minimum number of searchers that has the strategy to catch the target in the simultaneous move model. We showed that the number is as big as $\lfloor \frac{n}{2} \rfloor + 1$ when the target is at least as fast as the searchers, and as $\lfloor \frac{n}{v+1} \rfloor + 1$ when the searchers are $v > 1$ times faster than the target.

References

1. Adler, M., Räcke, H., Sivadasan, N., Sohler, C., Vöcking, B.: Randomized Pursuit-Evasion in Graphs. Combinatorics, Probability & Computing 12, 225–244 (2003)
2. Ahlswede, R., Bezrukov, S.L.: Edge Isoperimetric Theorems for Integer Point Arrays. Applied Mathematics Letters 8, 75–80 (1995)
3. Bezrukov, S.L.: Construction of the Solutions of a Discrete Isoperimetric Problem in a Hamming Space. Mathematics USSR-Sbornik 63, 81–96 (1989)
4. Bollobas, B., Leader, I.: An Isoperimetric Inequality on the Discrete Torus. SIAM Journal on Discrete Mathematics 3, 32–37 (1990)
5. Dumitrescu, A., Suzuki, I., Zylinski, P.: Offline Variants of the "Lion and Man" Problem. In: SoCG 2007. Proc. 23rd Annual Symposium on Computational Geometry, pp. 102–111. ACM Press, New York (2007)
6. Efrat, A., Guibas, L.J., Har-Peled, S., Lin, D.C., Mitchell, J.S.B., Murali, T.M.: Sweeping a Polygon with a Chain of Guards. In: SODA 2000 (Proc. 11th ACM-SIAM Symposium on Discrete Algorithms), pp. 927–936. ACM-Press, New York (2000)
7. Gal, S.: Search Problems with Mobile and Immobile Hider. SIAM Journal on Control and Optimization 17, 99–122 (1979)
8. Goldstein, A.S., Reingold, E.M.: The Complexity of Pursuit on a Graph. Theoretical Computer Science 143, 93–112 (1995)
9. Guibas, L.J., Latombe, J.-C., LaValle, S.M., Lin, D., Motwani, R.: A Visibility-Bases Pursuit-Evasion Problem. International Journal of Computational Geometry and Applications 9, 471–493 (1999)
10. Harary, F., Harborth, H.: Extremal Animals. Journal of Combinatorics, Information & System Science 1, 1–8 (1976)
11. Park, S.-M., Lee, J.-H., Chwa, K.-Y.: Visibility-Based Pursuit-Evasion in a Polygonal Region by a Single Searcher. In: Orejas, F., Spirakis, P.G., van Leeuwen, J. (eds.) ICALP 2001. LNCS, vol. 2076, pp. 456–468. Springer, Heidelberg (2001)
12. Parsons, T.D.: Pursuit-Evasion in a Graph. In: Alavi, Y., Lick, D. (eds.) Theory and Application of Graphs. Proc. International Conference Kalamazoo 1976. LNM, pp. 426–441. Springer, Heidelberg (1978)
13. Parsons, T.D.: The Search Number of a Connected Graph, Congressus Numerantium. In: Proc. 9th Southeastern Conference on Combinatorics, Graph Theory, and Computing, vol. 21, pp. 549–554 (1978)
14. Sgall, J.: Solution to David Gale's Lion and Man Problem. Theoretical Computer Science 259, 663–670 (2001)
15. Suzuki, I., Yamashita, M.: Searching for a Mobile Intruder in a Polygonal Region. SIAM Journal on Computing 21, 863–888 (1992)

Geometric Spanner of Segments*

Yang Yang[1], Yongding Zhu[1], Jinhui Xu[1,**], and Naoki Katoh[2,***]

[1] Department of Computer Science and Engineering
State University of New York at Buffalo
Buffalo, NY 14260, USA
{yyang6,yzhu3,jinhui}@cse.buffalo.edu
[2] Department of Architecture and Architectural Systems
Kyoto University, Japan
naoki@archi.kyoto-u.ac.jp

Abstract. Geometric spanner is a fundamental structure in computational geometry and plays an important role in many geometric networks design applications. In this paper, we consider a generalization of the classical geometric spanner problem (called segment spanner): Given a set S of disjoint 2-D segments, find a spanning network G with minimum size so that for any pair of points in S, there exists a path in G with length no more than t times their Euclidean distance. Based on a number of interesting techniques (such as weakly dominating set, strongly dominating set, and interval cover), we present an efficient algorithm to construct the segment spanner. Our approach first identifies a set of Steiner points in S, then construct a point spanner for them. Our algorithm runs in $O(|Q| + n^2 \log n)$ time, where Q is the set of Steiner points. We show that Q is an $O(1)$-approximation in terms of its size when S is relatively "well" separated by a constant. For arbitrary rectilinear segments under L_1 distance, the approximation ratio improves to 2.

1 Introduction

In this paper, we consider the following generalization of the classical geometric spanner problem: Given a set O of n disjoint objects in Euclidean space and a constant $t > 1$, construct a graph G for O of minimum size so that for any pair of points $p_i \in o_i$ and $p_j \in o_j$, there exists a path $P(p_i, p_j)$ in G whose total length is at most $t \times d(p_i, p_j)$, where o_i and o_j are objects in O and $d(p_i, p_j)$ is the Euclidean distance between p_i and p_j. The path $P(p_i, p_j)$ consists of three parts, P_1, P_2 and P_3, where P_1 and P_3 are the portions of $P(p_i, p_j)$ inside o_i and o_j respectively. We assume that there implicitly exists an edge (or path) between any pair of points inside each object $o \in O$. Thus, the objective of minimizing

* The research of this work was supported in part by National Science Foundation (NSF) through a CAREER award CCF-0546509 and a grant IIS-0713489.
** Corresponding author.
*** The research of this author was supported by the project New Horizons in Computing, Grant-in-Aid for Scientific Research on Priority Areas,MEXT Japan.

T. Tokuyama (Ed.): ISAAC 2007, LNCS 4835, pp. 75–87, 2007.

the size of G is equivalent to minimizing the total number of vertices, and edges between vertices in different objects. In this paper, we consider the case where all objects are disjoint 2-D line segments.

Spanner is a fundamental structure in computational geometry and finds applications in many different areas. Extensive researches have been done on this structure and a number of interesting results have been obtained [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]. Almost all previous results consider the case in which the objects are points and seek to minimize the spanner's construction time, size, weight, maximum degree of vertex, diameter, or combination of them.

A common approach for constructing geometric spanner is the use of Θ-graph [1,2,3,4]. In [5], Arya *et al.* showed that a t-spanner with constant degree can be constructed in $O(n \log n)$ time. In [6,7], they gave a randomized construction of a sparse t-spanner with expected spanner diameter $O(\log n)$. In [9,10], Das *et al.* proposed an $O(n \log^2 n)$-time greedy algorithm for a t-spanner with $O(n)$ edges and $O(1)wt(MST)$ weight in 3-D space. Gudmundsson *et al.* showed in [11] that an $O(n)$ edges and $O(1)wt(MST)$ weight t-spanner is possible to be constructed in $O(n \log n)$ time.

In graph settings, Chandar *et al.* [8] showed that for an arbitrary positive edge-weighted graph G and any $t > 1, \epsilon > 0$, a t-spanner of G with weight $O(n^{\frac{2+\epsilon}{t-1}})wt(MST)$ can be constructed in polynomial time. They also showed that $(\log^2 n)$-spanners of weight $O(1)wt(MST)$ can be constructed.

For geometric spanners of objects other than points, Asano *et al.* considered the problem of constructing a spanner graph for a set of axis-aligned rectangles using rectilinear bridges and under L_1 distance [12]. They showed that in general it is NP-hard to minimize the dilation, and when the spanner graph is restricted to be trees with rectilinear edges, the problem can be solved using a linear program. They also considered other simple graphs such as paths and sorted paths, and presented polynomial time solution for each of them.

The spanner of segments problem considered in this paper is motivated by several interesting applications. One of them is for constructing bridges between a set of buildings so that the path (traveling through bridges) between locations in different buildings is close to their Euclidean distance [12]. Another application appears in wireless mesh networks. In such networks, a set of wireless routers (or stations) are to be installed in objects, such as streets or highways, so that for any pair of wireless devices in those objects there exists a routing path for them with length close to their Euclidean distance. The rationale of such distance requirement is for minimizing the total energy used for routing messages between them, as the energy consumption is proportional to the path length.

To build a spanner of segments, we view the construction as a two-phase process. In the first phase, a set of points (called Steiner points) are selected from each segment, and in the second phase, a spanner is constructed for the set of Steiner points. Since the second phase can be completed by using existing spanner algorithms (for points), our focus in this paper is thus on the first phase. Furthermore, since most existing spanners are sparse graphs (i.e. consist of $O(n)$ edges), minimizing the size of the segment spanner is equivalent to

minimizing the total number of Steiner points. Our objective is hence to obtain a spanner with a minimum number of Steiner points. Using a number of interesting techniques, we show that there exists an $O(|Q| + n^2 \log n)$ algorithm to identify a set Q of Steiner points whose size is an $O(1)$-approximation of the optimal size. Further, when the segments are rectilinear and under L_1 distance, the approximation ratio improves to 2. Due to the space limit, we omit a lot of details and proofs from this extended abstract.

2 Preliminaries

Let $S = \{s_1, s_2, \cdots, s_n\}$ be a set of n disjoint segments on a plane with each segment $s_i = \overline{a_i b_i}$, where a_i and b_i are the left and right endpoints of s_i respectively (For a vertical segment, a_i is the lower endpoint and b_i is the upper endpoint). A t-spanner G_S of S is a network which connects the segments in S and satisfies the following condition. For any two points p_i and p_j on segments s_i and s_j in S, there exists a path (called spanner path) of G_S between p_i and p_j and with length no more than $t \times |p_i p_j|$, where t is the *stretch factor* of the spanner and $|p_i p_j|$ is the Euclidean distance between p_i and p_j. The spanner G_S contains two types of line segments: input segments and segments connecting the input segments. We call the former *segments* and the latter *bridges* to distinguish them. The intersections of segments and bridges are called *Steiner* points.

As mentioned in previous section, our main objective for the spanner G_S is to minimize its size. The size of G_S is the sum of the number of vertices and edges. The vertices of G_S include all endpoints of the input segments and the Steiner points. The edges consist of bridges and subsegments fragmented by the Steiner points. For a segment s_i with k Steiner points, the number of subsegments on s_i is bounded by $O(k)$ (i.e. at most $k + 1$). Thus, to minimize the size of G_S, it is sufficient to minimize the total number of Steiner points and bridges.

To simplify the optimization task, our main idea is to separate the procedures of minimizing i) the number of Steiner points; and ii) the number of bridges. We consider the following approach: (1) compute a set Q of Steiner points with small size, then (2) construct a spanner G_Q for Q to minimize the number of bridges. The spanner G_Q together with the subsegments on S forms a spanner for S.

For a pair of segments s_i, $s_j \in S$, the distance between them is defined as $d(s_i, s_j) = \min_{p_i \in s_i, p_j \in s_j} |p_i p_j|$. The distance from s_i to S is defined as $d_i = \min_{j \neq i, s_j \in S} d(s_i, s_j)$. Let l_i be the length of s_i. The *relative separation ratio* of s_i in S is defined as d_i/l_i and the relative separation ratio of S is $\min_{s_i \in S} d_i/l_i$. In this paper, we assume that S is "well" separated in a sense that its relative separation ratio is no less than ϵ for some constant $\epsilon > 0$. The rationale of this assumption is that in wireless networks, if two segments are too close to each other, they can share a set of routers (or stations) and therefore can be viewed as one segment.

3 Minimizing the Number of Steiner Points

3.1 Weakly Dominating Set

To investigate how the Steiner points affect each other, we consider the problem of placing Steiner points on two disjoint segments, $s_1 = \overline{a_1 b_1}$ and $s_2 = \overline{a_2 b_2}$. Let p_1 and p_2 be two arbitrary points on s_1 and s_2 respectively. Let q_1 and q_2 be two Steiner points on the neighborhoods of p_1 and p_2 respectively such that the path $p_1 \rightarrow q_1 \rightarrow q_2 \rightarrow p_2$ is a spanner path for p_1 and p_2. In this case, we say that q_1 and q_2 *t-dominate* the pair of p_1 and p_2. Clearly, the positions of q_1 and q_2 are constrained by p_1 and p_2. If we fix p_1, p_2, and one Steiner point q_1, then all possible positions of the other Steiner point q_2 form a (possibly empty) interval $I_S(p_1, p_2, q_1)$ (which is a function of p_1, p_2, and q_1) on s_2 (see Figure 1).

For each pair of segments, our main idea is to isolate their Steiner point determination from that of the rest of the segments. As discussed above, for an arbitrary pair of points $p_1 \in s_1$ and $p_2 \in s_2$, the positions of their t-dominating pair q_1 and q_2 are constrained. To relax this constraint, when placing Steiner points on s_1, we assume that q_2 can be placed at arbitrary position on s_2. Ideally, we assume that q_2 always overlaps with p_2. Thus, we only need to consider the relation between q_1 and p_1. We say that q_1 *t-weakly dominates* p_1 and p_2 if the length of the path $p_1 \rightarrow q_1 \rightarrow p_2$ is no more than $t \times |p_1 p_2|$. If q_1 t-weakly dominates p_1 and p_2 for every possible choice of p_2 (while fixing p_1), then we say q_1 t-weakly dominates p_1. Our objective is thus to select a minimum number of points on s_1 so that every point on s_1 is t-weakly dominated by some selected Steiner point. We call such a set of points as a *t-weakly dominating set* of s_1.

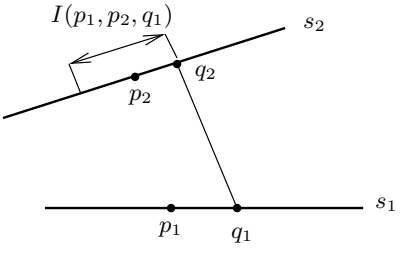


Fig. 1. Steiner points on two segments

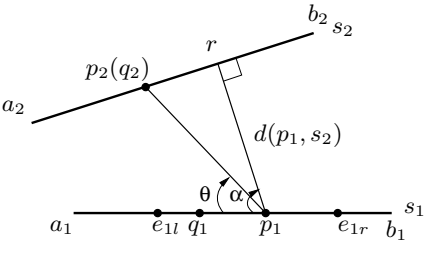


Fig. 2. The interval of $I(p_2, p_1)$

3.2 Computing *t*-Weakly Dominating Set in a Brute-Force Manner

Let $\theta = \angle a_1 p_1 p_2$, and e_{1l} and e_{1r} be the two endpoints of $I_S(p_2, p_1, p_2)$, i.e. the interval of all possible positions of q_1 while q_2 coincides with p_2 (see Figure 2).

Lemma 1. *The two endpoints e_{1l} and e_{1r} locate on different sides of p_1 with* $|p_1 e_{1l}| = \min\{|p_1 a_1|, \frac{t^2-1}{2(t-\cos\theta)}|p_1 p_2|\}$ *and* $|p_1 e_{1r}| = \min\{|p_1 b_1|, \frac{t^2-1}{2(t+\cos\theta)}|p_1 p_2|\ \}$.

Lemma 2. *$|p_1e_{1l}|$ (or $|p_1e_{1r}|$) achieves its minimum either when e_{1l} coincides with a_1 (or e_{1r} coincides with b_1), or p_2 is at the endpoints of s_2, or θ is one of the two constants depending only on s_1, s_2 and t.*

Proof. Let $d(p_1, s_2)$ be the shortest Euclidean distance from p_1 to s_2's supporting line and r be the point that achieves the shortest distance. Let α be the angle $\angle a_1p_1r$. See Figure 2 for an example. Since $|p_1p_2| = \frac{d(p_1,s_2)}{\cos(\alpha-\theta)}$, we have $|p_1e_{1l}| = \min\{|p_1a_1|, \frac{(t^2-1)d(p_1,s_2)}{2(t-\cos\theta)\cos(\alpha-\theta)}\}$. When p_1, s_2 and s_1 are fixed, both $d(p_1, s_2)$ and α are fixed. To achieve the minimum, $f_l(\theta) = (t - \cos\theta)\cos(\alpha - \theta)$ has to be maximized. $f_l(\theta)$ achieves its maximum either when θ is the root of $f_l'(\theta) = 0$ or θ is the minimum or maximum in its domain (i.e. when p_2 is at the endpoints of s_2), where $f_l'(\theta) = \sin(2\theta - \alpha) + t\sin(\alpha - \theta)$ is the derivative of $f_l(\theta)$. The equation of $f_l'(\theta) = 0$ has at most four roots, among which at most two are real roots that allow $f_l(\theta)$ to achieve the maximum for all p_1. W.l.o.g let θ_1, θ_2 be the two roots, and $f_l(\theta_1) \geq f_l(\theta_2)$ (See Figure 3 for an illustration). The lemma follows from the fact that the roots depend only on t and α. □

Lemma 3. *The set of t-weakly dominating points selected by the following procedure has the minimum size among all sets of points t-weakly dominating s_1.*

1. Mark all points on s_1 as non-dominated.
2. Starting from the first non-dominated point on s_1 (initially it is a_1), walk along s_1 until encounter the first point p_i, whose interval $I(p_i)$ overlaps at only one point, say q_i, with the common intersection of the intervals of all visited but non-dominated points.
3. Select q_i as a weakly dominating point, and mark all points visited in Step 2 as dominated points.
4. Keep walking along s_1 and marking points as dominated until the encountered point cannot be dominated by q_i.
5. Repeat Steps 2-4 until all points are dominated.

3.3 Parameterization

Let m be the parameter of p_1 in its convex combination of the two endpoints of s_1, i.e. $p_1 = (1 - m)a_1 + mb_1$, for $m \in [0, 1]$. Let $L_{1,2}(m)$ and $R_{1,2}(m)$ be the functions defining the positions of e_{1l} and e_{1r} (respectively) on s_1, i.e. $L_{1,2}(m) = m - |p_1e_{1l}|/|a_1b_1|$ and $R_{1,2}(m) = m + |p_1e_{1r}|/|a_1b_1|$.

Consider the two functions when m increases from 0 to 1, it is possible that the beginning part of $L_{1,2}(m)$ always has value 0 (because $e_{1l} = a_1$), and the ending part of $R_{1,2}(m)$ always has value 1 (because $e_{1r} = b_1$). To simplify the discussion in this section, from now on we focus on the remaining part of the two functions, and assume that $|p_1e_{1l}| = \frac{t^2-1}{2(t-\cos\theta)}|p_1p_2|$, $|p_1e_{1r}| = \frac{t^2-1}{2(t+\cos\theta)}|p_1p_2|$.

By Lemma 2, for each fixed m, $L_{1,2}(m)$ (or $R_{1,2}(m)$) is the maximum (or minimum) of $O(1)$ values with each corresponding to the position of e_{1l} (or e_{1r}) at a fixed θ value. Let $\Theta = \{\theta_1, \theta_2\}$ be the real roots of $f_l'(\theta) = 0$ (or $f_r'(\theta) = 0$) that allow $f_l(\theta)$ (or $f_r(\theta)$) to achieve its maximum. Since Θ depends only on the input

segments and t, it is the same for any $p_1 \in s_1$ and can be computed in advance. Let $\theta_a(m)$ and $\theta_b(m)$ be the two angles $\angle a_1 p_1 a_2$ and $\angle a_1 p_1 b_2$ respectively (i.e. when p_2 is at the two endpoints of s_2). By our definition, $\theta_b(m) \geq \theta_a(m)$, thus we have $\frac{t^2-1}{2(t-\cos\theta_b(m))} \leq \frac{t^2-1}{2(t-\cos\theta_a(m))}$ and $\frac{t^2-1}{2(t+\cos\theta_b(m))} \geq \frac{t^2-1}{2(t+\cos\theta_a(m))}$. Therefore, the position of e_{1l} depends on $\theta_b(m)$ and that of e_{1r} depends on $\theta_a(m)$.

$L_{1,2}(m)$ (or $R_{1,2}(m)$) can be viewed as the the upper (or lower) envelope of up to three functions, $g_i^l(m)$ (or $g_i^r(m)$), $1 \leq i \leq 2$, and $h^l(m)$ (or $h^r(m)$), where $g_i^l(m)$ (or $g_i^r(m)$) is the function of e_{1l} (or e_{1r}) when $\theta = \theta_i \in \Theta$, $h^l(m)$ is the function of e_{1l} when $\theta = \theta_b(m)$, $h^r(m)$ is the function of e_{1r} when $\theta = \theta_a(m)$.

Lemma 4. *Each $g_i^l(m)$ (or $g_i^r(m)$), $1 \leq i \leq 2$, is a linear function of m.*

Lemma 5. *Each of $h^l(m)$ and $h^r(m)$) is either a monotone function or the concatenation of a monotonically increasing function and a monotonically decreasing function.*

Proof. Let r_0 be the foot of perpendicular from b_2 to s_1, and m_0 be the parameter of r_0 in its affine combination of a_1 and b_1. Then we have $h^l(m) = m - \frac{t^2-1}{2(t-\cos\theta_b(m))}\frac{|p_1p_2|}{|a_1b_1|} = m - \frac{(t^2-1)(m-m_0)}{2(t-\cos\theta_b(m))\cos\theta_b(m)}$. Its derivative is

$$(h^l(m))' = \frac{2t^2\cos^2\theta_b(m) - (t^3+3t)\cos\theta_b(m) + t^2 + 1}{2(t-\cos\theta_b(m))^2}.$$

Let $x = \cos\theta_b(m)$. We get $\frac{2t^2x^2-(t^3+3t)x+t^2+1}{2(t-x)^2} = t^2 - \frac{3t^3-3t}{2(t-x)} + \frac{(t^2-1)^2}{2(t-x)^2} = \frac{1}{2}(\frac{t^2-1}{t-x} - t)(\frac{t^2-1}{t-x} - 2t)$. The function has two roots, $x_1 = \frac{1}{t}$, $x_2 = \frac{1}{2}(t+\frac{1}{t})$. Since $0 < x_1 < 1$, $x_2 > 1$, obviously x_2 is not feasible as $-1 \leq x = \cos\theta_b(m) \leq 1$. The derivative of $h^l(m)$ is decreasing on x when $x \leq \frac{1}{t}$; increasing otherwise. This shows that the function $h^l(m)$ can be partitioned into at most two pieces, with the first one monotonically increasing and the second monotonically decreasing on $\theta_b(m)$. □

Lemma 6. *$h^l(m)$ is an increasing function on $\theta_b(m)$ when $\theta_b(m) \geq \pi/2$; $h^r(m)$ is a decreasing function on $\theta_a(m)$ when $\theta_a(m) \leq \pi/2$. Further, the derivatives of $h^l(m)$ and $h^r(m)$ both have a minimum value of $\frac{1}{2}(1 + \frac{1}{t^2})$.*

3.4 Truncating the *h* Functions

Lemma 5 shows that each of $h^l(m)$ and $h^r(m)$ could be a bitonic function (i.e. an increasing function followed by a decreasing function). Taking $h^l(m)$ as an example, the geometric meaning of this lemma is that, as p_1 moves along s_1 from left to right, the left endpoint of $I(p_1, s_2)$ calculated by fixing p_2 at b_2 might move from right to left at some positions.

In such a scenario, let p_i be the first such point on s_1 that the left endpoint e_{il} of interval $I(p_i, s_2)$ starts to move "from right to left". By Lemma 5, every point p_j on s_1 that is to the right of p_i will have its e_{jl} (calculated by the same function $h^l(m)$) located to the left of e_{il}. This means that if we replace e_{jl} by e_{il} for every such p_j, we can make $h^l(m)$ a completely monotone function. Geometrically, this seems to truncate the tail of $h^l(m)$ (i.e. the decreasing piece) and replace it with a constant function. See Figure 4.

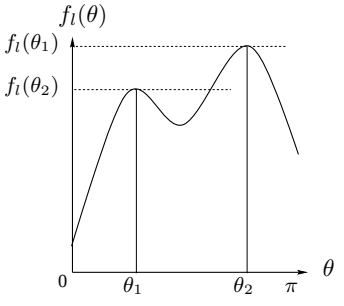


Fig. 3. An illustration of $f_l(\theta)$

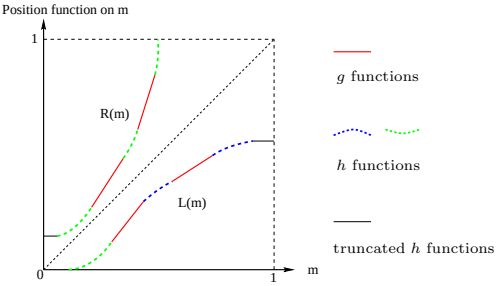


Fig. 4. An illustration of $L_{1,2}(m)$ and $R_{1,2}(m)$

3.5 Computing t-Weakly Dominating Set

The two functions $L_{1,2}(m)$ and $R_{1,2}(m)$ together form a "band" $B_{1,2}$ (see Figure 4). A horizontal line segment located within the band represents the interval $I(p_1, s_2)$ of a point p_1 on s_1 determined by the corresponding e_{1l} and e_{1r} positions. An interval cover $IC_{1,2}$ for $B_{1,2}$ is a set of horizontal intervals inside $B_{1,2}$ so that the union of each interval's vertical projection covers the domain of m. It is easy to see that an interval cover for $B_{1,2}$ corresponds to a t-weakly dominating set for s_1. This is because each horizontal interval uniquely determines a point on s_1. Since every point on s_1 is covered by some interval, it is t-weakly dominated by the point corresponding to that interval.

Let $L(m) = \{l_1, l_2, \ldots, l_{n_1}\}$ be the 2-D curve corresponding to the function of $L_{1,2}(m)$, where each segment (l_i, l_{i+1}) corresponds to a piece of the h or g function in $L_{1,2}(m)$. Similarly we have $R(m) = \{r_1, r_2, \ldots, r_{n_2}\}$.

Lemma 7. *The following algorithm computes an interval cover of minimum size in $O(|IC| + n_1 + n_2)$ time, where $|IC|$ is the size of the interval cover and n_1 and n_2 are the number of vertices in $L(m)$ and $R(m)$ respectively.*

1. Starting from r_1, shoot a horizontal ray to the right, and let v_1 be the intersection of the ray and $L(m)$.
2. Shoot a vertical ray upwards from v_1, and let v_2 be the intersection of the ray with $R(m)$.
3. Starting from v_2, repeat the above steps until all the $L(m)$ and $R(m)$ are horizontally covered.
4. Return the set of the horizontal segments as the cover. See Figure 5.

3.6 Imaginary Steiner Points

Let p_1, p_2 be two arbitrary points on segments s_1 and s_2 respectively, and q_1, q_2 be their t-weakly dominating points. Ideally, the path $p_1 \to q_1 \to q_2 \to p_2$ should be a t-spanner path for p_1, p_2, i.e. $|p_1q_1|+|q_1q_2|+|q_2p_2| \leq t|p_1p_2|$. Due to the weak domination, we know $|p_1q_1| + |q_1q_2| \leq t|p_1q_2|$, $|p_2q_2| + |q_2q_1| \leq t|p_2q_1|$. Adding

the two together, we have $|p_1q_1| + |q_1q_2| + |q_2p_2| \leq t|p_1q_2| + t|p_2q_1| - |q_1q_2|$. To make q_1 and q_2 t-dominate p_1 and p_2, we need $t|p_1q_2| + t|p_2q_1| - |q_1q_2| \leq t|p_1p_2|$.

Our main idea is to introduce an *imaginary* Steiner point p_M, which is the median of $\overline{p_1p_2}$, and use this imaginary Steiner point to help determining the dominating points for p_1 and p_2 (see Figure 6). More specifically, when computing the interval $I(p_1, s_2)$, we assume that there exists a Steiner point p_M at the median of $\overline{p_1p_2}$ for every possible choice of p_2. These imaginary Steiner points form an imaginary "Steiner" segment s_2' for every p_1 (See Figure 6 for an example). Instead of computing $I(p_1, s_2)$ directly, we calculate $I(p_1, s_2') = \cap_{p_M \in s_2'} I_S(p_M, p_1, p_M)$. ($I(p_2, s_1')$ can be defined similarly.) Steiner point q_1 in such $I(p_1, s_2')$ is therefore a t-weakly dominating point for p_1 and $\forall p_M \in s_2'$.

Lemma 8. *Let $q_1 \in I(p_1, s_2')$ and $q_2 \in I(p_2, s_1')$ be t-weakly dominating Steiner points for p_1 and p_2 with respect to their imaginary Steiner segments respectively. Then, q_1 and q_2 are a t-dominating pair for p_1 and p_2.*

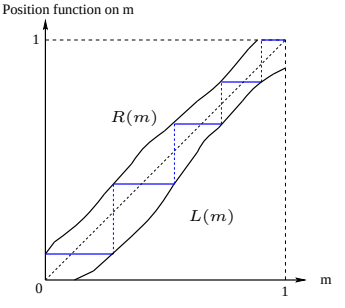


Fig. 5. An example of the interval cover

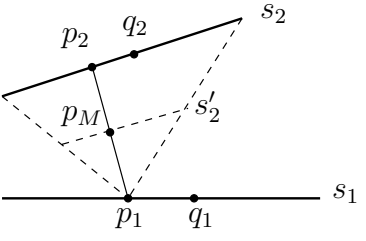


Fig. 6. Imaginary Steiner point p_M

3.7 From Weakly Dominating Set to Dominating Set

Let m_1 be the parameter of p_1 when e_{1l} is at a_1 (i.e. $L_{1,2}(m_1) = 0$ and $\forall m_1 < m \leq 1$, $L_{1,2}(m) > 0$), and m_2 be the parameter of p_1 when e_{1r} coincides with b_1 (i.e. $R_{1,2}(m_2) = 1$ and $\forall 0 \leq m < m_2$, $R_{1,2}(m) < 1$).

Lemma 9. *$\bar{L}_{1,2}(m) = (m + L_{1,2}(m))/2$ for $m_1 \leq m \leq 1$; $\bar{R}_{1,2}(m) = (m + R_{1,2}(m))/2$ for $0 \leq m \leq m_2$.*

Let H_L be the longest maximal horizontal line segment within $B_{1,2}$, and H_S be the shortest maximal horizontal line segment within $B_{1,2}$.

Lemma 10. *$|H_L|/|H_S| \leq \frac{1}{(t-1)\epsilon}$, where ϵ is the relative separation ratio of S.*

Proof. A horizontal line segment within $B_{1,2}$ satisfying $R_{1,2}(\lambda_1) = L_{1,2}(\lambda_{1'})$ represents a t-weakly dominating point q determined by two points p_1 and $p_{1'}$ on s_1 (corresponding to parameter λ_1 and $\lambda_{1'}$ respectively, see Figure 7 for an example.). For p_1, assume that the point on the segment s_2 that allows

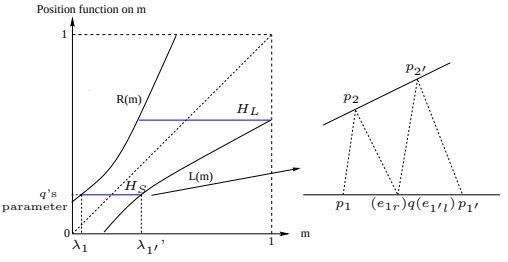


Fig. 7. An example of H_L and H_S

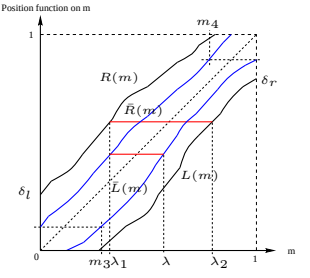


Fig. 8. An example of $B_{1,2}$ and $\bar{B}_{1,2}$

e_{1r} to achieve its minimum is p_2. For $p_{1'}$, assume that the point on s_2 that allows $e_{1'l}$ to achieve its minimum is $p_{2'}$. Since q is the t-weakly dominating point determined by such minimum p_1e_{1r} and $p_{1'}e_{1'l}$, they (q, e_{1r} and $e_{1'l}$) coincide at one point. Then we have $|p_1e_{1r}| + |p_2e_{1r}| = |p_1q| + |p_2q| = t|p_1p_2|$ and $|p_{1'}e_{1'l}|+|p_{2'}e_{1'l}| = |p_{1'}q|+|p_{2'}q| = t|p_{1'}p_{2'}|$ (equalities are achieved because p_1e_{1r} and $p_{1'}e_{1'l}$ are minimum). By triangle inequality, $|p_2q| \leq |p_1q| + |p_1p_2|$, $|p_{2'}q| \leq |p_{1'}q| + |p_{1'}p_{2'}|$. Hence, $|p_1q| + |p_{1'}q| \geq (t-1)(|p_1p_2| + |p_{1'}p_{2'}|)/2$. As we described before, the segments in S are well separated with relative separation ratio ϵ, therefore $|p_1p_2| \geq \epsilon|a_1b_1|$ and $|p_{1'}p_{2'}| \geq \epsilon|a_1b_1|$. Thus we have $|H_S| = (|p_1q| + |p_{1'}q|)/|a_1b_1| \geq (t-1)\epsilon$. The lemma follows since $|H_L| \leq 1$. □

$\bar{L}_{1,2}(m)$ and $\bar{R}_{1,2}(m)$ form a "shrunk" band $\bar{B}_{1,2}$. A minimum-sized interval cover $\overline{IC}_{1,2}$ can also be found within $\bar{B}_{1,2}$ by using the algorithm given in Section 3.5. Let $\beta = |\overline{IC}_{1,2}|/|IC_{1,2}|$. Let $\delta_l = R_{1,2}(0)$ and $\delta_r = L_{1,2}(1)$. By Lemma 9, we have $\bar{R}_{1,2}(0) = \delta_l/2$ and $\bar{L}_{1,2}(1) = (1+\delta_r)/2$. Let m_3 and m_4 be the parameter of p_1 satisfying $\bar{L}_{1,2}(m_3) = \delta_l/2$ and $\bar{R}_{1,2}(m_4) = (1+\delta_r)/2$ respectively.

Lemma 11. *If $\delta_r \geq \delta_l$, then $\beta \leq \dfrac{1}{(t-1)\epsilon} \cdot \min\{1 + 1/(2 \min_{m_3<m<1} \bar{L}'_{1,2}(m) - 1),$ $1 + 1/(2 \min_{0<m<m_4} \bar{R}'_{1,2}(m) - 1)\}$, where $\bar{L}'_{1,2}(m)$ and $\bar{R}'_{1,2}(m)$ are the derivatives of $\bar{L}_{1,2}(m)$ and $\bar{R}_{1,2}(m)$ respectively.*

Proof. Let H be any maximal horizontal line segment within $B_{1,2}$ with endpoints $(\lambda_1, R_{1,2}(\lambda_1))$ and $(\lambda_2, L_{1,2}(\lambda_2))$. Obviously, $R_{1,2}(\lambda_1) = L_{1,2}(\lambda_2)$. Starting at point $(\lambda_1, \bar{R}_{1,2}(\lambda_1))$, there exists a maximal horizontal line segment $\bar{H}$ within $\bar{B}_{1,2}$ with right endpoint $(\lambda, \bar{L}_{1,2}(\lambda))$. Clearly,$\lambda \in [\lambda_1, \lambda_2]$ and $\bar{R}_{1,2}(\lambda_1) = \bar{L}_{1,2}(\lambda)$. Assume that we can extend $L_{1,2}(m)$ such that Lemma 9 can be applied to all $m \in [0,1]$ (i.e. allow e_{1l} to be placed to the left of a_1; note that this will not affect bounding β). By Lemma 9, $(\lambda_1 + R_{1,2}(\lambda_1))/2 = (\lambda + L_{1,2}(\lambda))/2$, i.e. $R_{1,2}(\lambda_1) - L_{1,2}(\lambda) = \lambda - \lambda_1$. Since $R_{1,2}(\lambda_1) = L_{1,2}(\lambda_2)$, $L_{1,2}(\lambda_2) - L_{1,2}(\lambda) = \lambda - \lambda_1$. By Mean Value Theorem (the function $L_{1,2}(m)$ is smooth when $L_{1,2}(m) > 0$), $L'_{1,2}(\lambda')(\lambda_2 - \lambda) = \lambda - \lambda_1$, where $\lambda < \lambda' < \lambda_2$. Thus $|H|/|\bar{H}| = (\lambda_2 - \lambda_1)/(\lambda - \lambda_1) = 1 + (\lambda_2 - \lambda)/(\lambda - \lambda_1) = 1 + 1/L'_{1,2}(\lambda') = 1 + 1/(2\bar{L}'_{1,2}(\lambda') - 1)$. Consider each interval $\bar{H}_i \in \overline{IC}_{1,2}$, correspondingly there is a maximal horizontal segment H_i within $B_{1,2}$. If we consider

all $|\bar{H}_i|$ and find the minimum, say $|\bar{H}_i| = \tilde{\lambda} - \tilde{\lambda}_1$ with $\tilde{\lambda}_1 < \tilde{\lambda} < \tilde{\lambda}_2$, we have $|\overline{IC}_{1,2}| \leq \frac{1}{\tilde{\lambda}-\tilde{\lambda}_1}$. For the corresponding $|H_i|$, we have $|H_i| = \tilde{\lambda}_2 - \tilde{\lambda}_1 \geq |H_S|$. Consider each interval $H_j \in IC_{1,2}$, we have $|H_j| \leq |H_L|$ such that $|IC_{1,2}| \geq \frac{1}{|H_L|}$. Hence, $\beta = \frac{|\overline{IC}_{1,2}|}{|IC_{1,2}|} \leq \frac{|H_L|}{\tilde{\lambda}-\tilde{\lambda}_1} \leq \frac{\tilde{\lambda}_2-\tilde{\lambda}_1}{\tilde{\lambda}-\tilde{\lambda}_1} \times \frac{|H_L|}{|H_S|}$. By Lemma 10, $\beta \leq \max_{\lambda_1<\lambda<\lambda_2} \frac{(\lambda_2-\lambda_1)}{(\lambda-\lambda_1)} \times \frac{|H_L|}{|H_S|} \leq \frac{1}{(t-1)\epsilon}(1 + \frac{1}{2\min_{\lambda<\lambda'<\lambda_2} \bar{L}'_{1,2}(\lambda') - 1})$. Since $\bar{R}_{1,2}(0) = \bar{L}_{1,2}(m_3)$, $\beta \leq \frac{1}{(t-1)\epsilon}(1 + \frac{1}{2\min_{m_3<m<1} \bar{L}'_{1,2}(m) - 1})$. See Figure 8. □

Lemma 12. *$\beta \leq \frac{1}{(t-1)\epsilon} \times \max\{1 + \frac{2f_l(\theta_2)}{2f_l(\theta_2)-(t^2-1)\cos\alpha}, 1 + \frac{2f_r(\theta_2)}{2f_r(\theta_2)-(t^2-1)\cos\alpha}, 3 - \frac{2}{t^2+1}\}$. And β is $O(1)$ since $f_l(\theta_2)$, $f_r(\theta_2)$, t and ϵ are all constants.*

Proof. If $\delta_r \leq \delta_l$, one Steiner point is sufficient to t-weakly dominate s_1. Since the t-dominating interval of a point p_1 is always half of its t-weakly dominating interval, by choosing q, a_1 and b_1 as the Steiner points, they are sufficient to t-dominate s_1. In this case, $\beta \leq 3$. Thus, from now on we assume $\delta_r > \delta_l$.

1. For the parts of $L_{1,2}(m)$ determined by the linear g_i functions, we have
$$\min L'_{1,2}(m) = \min_{\theta_i \in \Theta}\{1 - \frac{(t^2-1)\cos\alpha}{2(t-\cos\theta_i)\cos(\alpha-\theta_i)}\} = 1 - \frac{(t^2-1)\cos\alpha}{2f_l(\theta_2)}.$$
2. For the parts of $L_{1,2}(m)$ determined by $h^l(m)$, $\theta_b(m)$ decreases when m increases. Let m_b be the parameter of p_1 such that $\theta_b(m_b) = \pi/2$. Then, $\theta_b(m) > \pi/2$ for $0 \leq m < \min\{m_b, m_2\}$, $\theta_b(m) < \pi/2$ for $\max\{0, m_b\} < m \leq 1$. (One of the intervals in the above two inequalities might be empty.)
   (a) $m \in [0, \min\{m_b, m_2\}]$, by Lemma 6 $\min L'_{1,2}(m) = \min \frac{dh^l}{dm} \geq \frac{1}{2}(1 + \frac{1}{t^2})$.
   (b) For $m \in [\max\{0, m_b\}, 1]$, we have two subcases.
      i. If $R_{1,2}(m)$ is determined by g_i, $\min R'_{1,2}(m) = 1 - \frac{(t^2-1)\cos\alpha}{2f_r(\theta_2)}$.
      ii. If $R_{1,2}(m)$ is determined by $h^r(m)$, we have $\theta_a(m) \leq \theta_b(m) \leq \pi/2$. By Lemma 6, $\min R'_{1,2}(m) = \min \frac{dh^r}{dm} \geq \frac{1}{2}(1 + \frac{1}{t^2})$.

Combining all the cases, the lemma follows from Lemma 11. □

3.8 From Dominating Set To Strongly Dominating Set

For any two segments s_i and s_j in S, s_i and s_j are weakly visible to each other if there exists a pair of points $p_i \in s_i$ and $p_j \in s_j$ such that p_i and p_j are visible to each other (i.e. $\overline{p_ip_j}$ does not intersect the interior of any other input segment).

Lemma 13. *To compute a t-strongly dominating set of an arbitrary segment s_1, it is sufficient to consider only those segments weakly visible to it.*

Let $WV_i = \{s_1, s_2, \ldots, s_{k_i}\}$ be the set of segments weakly visible to s_i.

1. For each segment $s_i \in S$, compute WV_i.
   (a) For each segment $s_j \in WV_i$, compute the g and h functions for s_i with respect to s_j.
   (b) Determine $L_{i,j}(m)$ and $R_{i,j}(m)$.
2. Let $L_i(m)$ be the upper envelope of the set of $L_{i,j}(m)$ functions, and $R_i(m)$ be the lower envelope of the set of $R_{i,j}(m)$ functions.
3. Determine $\bar{L}_i(m)$ and $\bar{R}_i(m)$ with the help of the imaginary Steiner points.
4. Compute an interval cover $\overline{IC}_i$ for the band formed by $\bar{L}_i(m)$ and $\bar{R}_i(m)$.
5. For each interval in $\overline{IC}_i$, determine its corresponding t-strongly dominating Steiner point.

Consider s_i and WV_i. For each pair of segments s_i and s_j, $s_j \in WV_i$, we have a pair of parameterized functions $L_i(m)$ and $R_i(m)$. It is not difficult to see that both $L_i(m)$ and $R_i(m)$ are piecewise smooth.

Lemma 14. *Let $\beta_{i[u,v]}$ be the ratio of $\overline{IC}_i$ to IC_i in the interval $[u, v]$. Then,*

$$\beta_{i[u,v]} \leq \frac{1}{(t-1)\epsilon} \times \min\{1 + 1/\min_{0<m<1} L_i(m), 1 + 1/\min_{0<m<1} R_i(m)\}.$$

Proof. Notice that if $L_{i,j}(\lambda) < L_{i,k}(\lambda)$, then $\bar{L}_{i,j}(\lambda) < \bar{L}_{i,k}(\lambda)$. Same property holds for the $R_{i,j}(m)$ functions. This property ensures that the ideas used in the proof of Lemma 11 can still be applied on each smooth piece. □

Lemma 15. *Let* $K_1 = \max_{\forall s_i, s_j \in S} \frac{2f_l(\theta_2)}{2f_l(\theta_2) - (t^2-1)\cos\alpha}$ *and* $K_2 = \max_{\forall s_i, s_j \in S} \frac{2f_r(\theta_2)}{2f_r(\theta_2) - (t^2-1)\cos\alpha}$. *Then,* $\beta_i \leq \frac{1}{(t-1)\epsilon} \times \max\{1 + K_1, 1 + K_2, 3 - \frac{2}{t^2+1}\}$.

Lemma 16. *The above algorithm computes a t-strongly dominating set for S in $O(|Q| + n^2 \log n)$ time, where $|Q|$ is the size of the t-strongly dominating set.*

Theorem 1. *For a set S of n disjoint 2-D segments with constant relative separation ratio, a set of t-strongly dominating Steiner points whose size is an $O(1)$-approximation of (the size of) the optimal solution can be computed in $O(|Q| + n^2 \log n)$ time, where $|Q|$ is the size of the set of Steiner points.*

4 Minimizing the Size of the Segment Spanner

Consider an optimal solution $\mathcal{O}$. Let N be the number of Steiner points in $\mathcal{O}$, and M be the number of bridges in $\mathcal{O}$. It is easy to see that $\mathcal{O}$ contains: i) $2n$ endpoints and N Steiner points; ii) $N+n$ subsegments and M bridges. Therefore $|\mathcal{O}| = 3n + 2N + M$. Recently, [13] shows that given a set of n_0 points S, in the worst case, any graph with $n_0 - 1 + k$ edges on S has dilation at least $\frac{2n_0}{\pi(k+1)}$. That is to say, as a t-spanner for the input segment, $\mathcal{O}$ needs to contain at least $M = 2N/\pi t + N - 2$ edges. Therefore $|\mathcal{O}| \geq 3n + 3N + 2N/\pi t - 2$.

Consider a spanner $\mathcal{A}$ generated by our algorithm. Let M' be the number of bridges in $\mathcal{A}$. $\mathcal{A}$ contains: i) $2n$ endpoints and βN Steiner points; ii) $\beta N + n$ subsegments and M' bridges. Therefore $|\mathcal{A}| = 3n+2\beta N+M'$. Again, in [13], the authors give an algorithm that for a set of n_0 points S, finds a spanner on S with at most n_0-1+k edges and dilation $O(\frac{n}{k+1})$. Thus if using the algorithm in [13], we have $t_2 = O(\frac{\beta N}{M'-\beta N+2}) = \frac{c\cdot\beta N}{M'-\beta N+2}$. Therefore $M' = c\cdot\beta N/t_2 + \beta N - 2$ and $|\mathcal{A}| = 3n + 3\beta N + c\cdot\beta N/t_2 - 2$. To achieve the best approximation ratio, we can minimize the ratio $\frac{|\mathcal{A}|}{|\mathcal{O}|} \leq \frac{3n+3\beta N+c\cdot\beta N/t_2-2}{3n+3N+2N/\pi t-2}$. Since β is a function of t_1 and $t_1t_2 = t$, we can choose t_1 and t_2 to minimize $|\mathcal{A}|/|\mathcal{O}|$.

5 Constructing t-Spanner for Rectilinear Segments Under L_1 Distance

Assume that the input is a set S of rectilinear segments, and the distance function is based on the L_1 distance (i.e. the Manhattan distance).

Theorem 2. *Given a set of n rectilinear segments, a set of t-strongly dominating set of Steiner points with size no more than $2 \times |OPT|$ can be computed in $O(|Q| + n^2 \log n)$ time, where $|Q|$ is the size of the set of Steiner points.*

Notice that in theorem 2, the segments are not required to be well separated.

References

1. Keil, J.M.: Approximating the complete euclidean graph. In: 1st Scandinavian Workshop on Algorithm Theory, pp. 208–213 (1988)
2. Keil, J.M., Gutwin, C.A.: Classes of graphs which approximate the complete euclidean graph. Discrete and Computational Geometry 7, 13–28 (1992)
3. Rupper, J., Seidel, R.: Approximating the d-dimensional complete euclidean graph. In: 3rd Canadian Conference on Computational Geometry, pp. 207–210 (1991)
4. Clarkson, K.L.: Approximation algorithms for shortest path motion planning. In: Proceedings of the nineteenth annual ACM conference on Theory of computing, pp. 56–65 (1987)
5. Arya, S., Das, G., Mount, D.M., Salowe, J.S., Smid, M.: Euclidean spanners: short, thin, and lanky. In: Proceedings of the twenty-seventh annual ACM symposium on Theory of computing, pp. 489–498 (1995)
6. Arya, S., Mount, D.M., Smid, M.: Dynamic algorithms for geometric spanners of small diameter: randomized solutions. Technical report, Max-Planck-Institut für Informatik (1994)
7. Arya, S., Mount, D.M., Smid, M.: Randomized and deterministic algorithms for geometric spanners of small diameter. In: 35th IEEE Symposium on Foundtions of Computer Science, pp. 703–712 (1994)
8. Chandra, B., Das, G., Narasimhan, G., Soares, J.: New spareness results on graph spanners. In: Proceedings of the eighth annual symposium on Computational geometry, pp. 192–201 (1992)

9. Das, G., Heffernan, P., Narasimhan, G.: Optimally sparse spanners in 3-dimensional euclidean space. In: Proceedings of the ninth annual symposium on Computational geometry, pp. 53–62 (1993)
10. Das, G., Narasimhan, G.: A fast algorithm for constructing sparse euclidean spanners. In: Proceedings of the tenth annual symposium on Computational geometry, pp. 132–139 (1994)
11. Gudmundsson, J., Levcopoulos, C., Narasimhan, G.: Fast greedy algorithms for constructing sparse geometric spanners. SIAM - Journal on Computing 31(5), 1479–1500 (2002)
12. Asano, T., de Berg, M., Cheong, O., Everett, H., Haverkort, H., Katoh, N., Wolff, A.: Optimal spanners for axis-aligned rectangles. Comput. Geom. Theory Appl. 30(1), 59–77 (2005)
13. Aronov, B., de Berg, M., Cheong, O., Gudmundsson, J., Haverkort, H., Smid, M., Vigneron, A.: Sparse geometric graphs with small dilation. In: Proceedings of the 12th Computing: The Australasian Theroy Symposium, vol. 51 (2006)

Dilation-Optimal Edge Deletion in Polygonal Cycles*

Hee-Kap Ahn[1], Mohammad Farshi[2,4,**], Christian Knauer[3], Michiel Smid[4,***], and Yajun Wang[5]

[1] Department of Computer Science and Engineering, POSTECH, Pohang, Korea
heekap@postech.ac.kr
[2] Department of Mathematics and Computing Science, TU Eindhoven, P.O. Box 513, 5600 MB Eindhoven, The Netherlands
m.farshi@tue.nl
[3] Institut für Informatik, Freie Universität Berlin, Takustraße 9, D–14195 Berlin, Germany
christian.knauer@inf.fu-berlin.de
[4] School of Computer Science, Carleton University, Ottawa, Ontario, K1S 5B6, Canada. michiel@scs.carleton.ca
[5] Department of Computer Science and Engineering, HKUST, Hong Kong S.A.R, China
yalding@cse.ust.hk

Abstract. Let C be a polygonal cycle on n vertices in the plane. A randomized algorithm is presented which computes in $O(n \log^3 n)$ expected time, the edge of C whose removal results in a polygonal path of smallest possible dilation. It is also shown that the edge whose removal gives a polygonal path of largest possible dilation can be computed in $O(n \log n)$ time. If C is a convex polygon, the running time for the latter problem becomes $O(n)$. Finally, it is shown that for each edge e of C, a $(1 - \epsilon)$-approximation to the dilation of the path $C \setminus \{e\}$ can be computed in $O(n \log n)$ total time.

1 Introduction

Given a (geometric) network, a natural question to ask is what happens to the quality of the network when some connections are removed. In case some links in a traffic network have to be shut down (e.g., due to budget considerations), we may want to know which edges of the network should be removed so as to not decrease the quality of the new network too much. Alternatively, we may want to know the most critical edge in the network, i.e., the edge whose removal causes the largest possible decrease in the quality of the new network. We consider a

* Part of this work was done during the Korean Workshop on Computational Geometry at Schloß Dagstuhl in 2006.
** This author was supported by Ministry of Science, Research and Technology of I. R. Iran.
*** This author was supported by NSERC.

T. Tokuyama (Ed.): ISAAC 2007, LNCS 4835, pp. 88–99, 2007.

simple variant of this problem: The initial network is a polygonal cycle C in the plane, and we have to remove one single edge from C. We measure the quality of the resulting polygonal path P by its *dilation* (or stretch factor) δ_P.

Recall that the dilation between two distinct *vertices* x and y of the path P is defined as $\delta_P(x,y) := d_P(x,y)/|xy|$, where $d_P(x,y)$ denotes the Euclidean length of the subpath of P connecting x and y, and $|xy|$ denotes the Euclidean distance between x and y. For convenience we define $\delta_P(x,x) := 1$. The dilation between two *sets* X and Y of *vertices* of P is defined as

$$\delta_P(X,Y) := \max\{\delta_P(x,y) \mid x \text{ is a vertex of } X,\ y \text{ is a vertex of } Y\},$$

the dilation of a *set* X of *vertices* of P is defined as $\delta_P(X) := \delta_P(X,X)$, and the *dilation* of the path P is defined as

$$\delta_P := \delta_P(P) = \max\{\delta_P(x,y) \mid x \text{ and } y \text{ are vertices of } P\}.$$

The problem we consider is the following: We are given a polygonal cycle $C = (p_0, \ldots, p_{n-1}, p_0)$ whose n vertices $p_0, \ldots, p_{n-1}$ are points in the plane. We want to determine the edge e of C for which the dilation of the polygonal path $C \setminus \{e\}$ is minimized or maximized. In other words, if we denote by P_i (for $0 \le i < n$) the polygonal path obtained by removing the edge (p_i, p_{i+1}) from C (where indices are to be read modulo n), then our goal is to compute $\delta_C^{\min} := \min_{0 \le i < n} \delta_{P_i}$ and $\delta_C^{\max} := \max_{0 \le i < n} \delta_{P_i}$. We will prove the following results :

Theorem 1. *Given a polygonal cycle C on n vertices in the plane, we can compute $\delta_C^{\min}$ in $O(n \log^3 n)$ expected time.*

Theorem 2. *Given a polygonal cycle C on n vertices in the plane, we can compute $\delta_C^{\max}$ in $O(n \log n)$ time. If C is a convex polygon, $\delta_C^{\max}$ can be computed in $O(n)$ time.*

Theorem 3. *Given a polygonal cycle $C = (p_0, \ldots, p_{n-1}, p_0)$ on n vertices in the plane and a constant $\epsilon > 0$, in $O(n \log n)$ time, we can compute a sequence $t_0, \ldots, t_{n-1}, t^*$ of real numbers, such that $\delta_{P_i}/(1+\epsilon) \le t_i \le \delta_{P_i}$ for each $i = 0, 1, \ldots, n-1$ and $\delta_C^{\min}/(1+\epsilon) \le t^* \le \delta_C^{\min}$.*

In Sect. 2, we prove Theorem 1. We start in Sect. 2.1 by describing an approach of [1] to estimate the dilation of a polygonal path; see also [7]. These ideas will play a central role in the algorithm we give in Sect. 2.2 for solving a decision problem associated with the problem of computing $\delta_C^{\min}$; the algorithm solving this decision problem runs in $O(n \log^2 n)$ expected time. In Sect. 2.3, we give a simple randomized approach which reduces the problem of computing $\delta_C^{\min}$ to an expected number of $O(\log n)$ decision problems of Sect. 2.2. Thus, this reduction incurs a logarithmic slowdown of the decision procedure.

In Sect. 3, we prove Theorem 2. We first show that for two fixed vertices x and y of C, it is easy to determine the largest possible dilation between them if one edge is removed from C. We then show that, in order to compute $\delta_C^{\max}$,

it suffices to consider pairs (x, y) of vertices whose distance is at most twice the closest-pair distance in the vertex set of C. Since there are only $O(n)$ such pairs (x, y), this leads to an efficient algorithm for computing $\delta_C^{\max}$.

Theorem 3 is proved in Sect. 4. The algorithm uses the well-separated pair decomposition of [2] and a result of [10], which states that this decomposition can be used to reduce the problem of approximating the dilation of a Euclidean graph to the problem of computing the shortest-path distances between $O(n)$ pairs of vertices. This result, together with the observation that for any two vertices x and y of C, the sequence $\delta_{P_0}(x, y), \ldots, \delta_{P_{n-1}}(x, y)$ contains only two distinct values, leads to an $O(n \log n)$–time algorithm that approximates the dilation of each path P_i as well as the minimum dilation $\delta_C^{\min}$.

2 Dilation-Minimal Edge Deletion in a Polygonal Cycle

2.1 Estimating the Dilation of a Polygonal Path

Our algorithm for computing the edge of a polygonal cycle whose removal minimizes the dilation of the resulting path uses as a subroutine parts of the algorithm of [1] that decides if the dilation of a polygonal path is less than some given threshold $\kappa > 1$; see also [7]. We describe those parts of this algorithm which are relevant for us.

Let $P = (p_0, \ldots, p_{n-1})$ be a polygonal path whose n vertices are points in the plane and let $\kappa \geq 1$ be a real number. The idea is to use a lifting transformation that rephrases the decision problem, i.e., the problem of deciding if $\delta_P < \kappa$, into a point-cone incidence-problem in $\mathbb{R}^3$.

We denote the first and last vertices of a polygonal path P by $f(P)$ and $l(P)$, respectively. Thus, $f(P) = p_0$. For each vertex p of P, we define the *weight* of p to be $\omega_P(p) := d_P(p, f(P))/\kappa$. We map each vertex $p = (x_p, y_p)$ of P to the point $h_P(p) := (x_p, y_p, \omega_P(p)) \in \mathbb{R}^3$. Let $\mathcal{C}$ denote the three-dimensional cone $\mathcal{C} := \{(x, y, z) \in \mathbb{R}^3 \mid z = \sqrt{x^2 + y^2}\}$. We map each vertex p of P to the cone

$$\mathcal{C}_P(p) := \mathcal{C} \oplus h_P(p) = \{c + h_P(p) \mid c \in \mathcal{C}\}.$$

If p and q are vertices of P, then we say that p is *before* q on P, if $d_P(p, f(P)) < d_P(q, f(P))$; this will be denoted as $p <_P q$. We then get the following lemma.

Lemma 1. *For any two vertices p and q of P with $p <_P q$, we have*

$$\delta_P(p, q) < \kappa \text{ if and only if } h_P(q) \text{ lies below } \mathcal{C}_P(p).$$

Proof. By straightforward algebraic manipulation, we have

$$\begin{aligned}
\delta_P(p, q) < \kappa &\iff \frac{d_P(q, p)}{|qp|} < \kappa \\
&\iff \frac{d_P(f(P), q) - d_P(f(P), p)}{|qp|} < \kappa \\
&\iff \frac{d_P(f(P), q)}{\kappa} < |qp| + \frac{d_P(f(P), p)}{\kappa} \\
&\iff \omega_P(q) < |qp| + \omega_P(p).
\end{aligned}$$

□

If X and Y are subsets of the vertex set of P, then we say that X is *before* Y on P, if $d_P(x, f(P)) < d_P(y, f(P))$ for all $x \in X$ and all $y \in Y$; this will be denoted as $X <_P Y$. For any subset X of the vertex set of P, we define $\mathcal{C}_P(X) := \{\mathcal{C}_P(p) \mid p \in X\}$ and $h_P(X) := \{h_P(p) \mid p \in X\}$.

The lower envelope of a set S of bi-variate functions will be denoted as $\mathcal{L}(S)$. Lemma 1 immediately gives the following result.

Lemma 2. *For any two subsets X and Y of the vertex set of P with $X <_P Y$, we have $\delta_P(X, Y) < \kappa$ if and only if $h_P(Y)$ lies below $\mathcal{L}(\mathcal{C}_P(X))$.*

The minimization diagram of $\mathcal{C}_P(X)$, i.e., the projection of the lower envelope $\mathcal{L}(\mathcal{C}_P(X))$ onto the xy-plane, is the additively weighted Voronoi diagram $V_P(X)$ of X with respect to the weight function ω_P. If the point y of Y is located in the Voronoi region of the point x of X, then $h_P(y)$ is below $\mathcal{L}(\mathcal{C}_P(X))$ if and only if $h_P(y)$ is below $\mathcal{C}_P(x)$.

This yields an efficient algorithm to verify if $\delta_P(X, Y) < \kappa$ for two subsets X and Y of the vertex set of P having the property that $X <_P Y$: The Voronoi diagram $V_P(X)$ can be computed in $O(|X| \log |X|)$ time, c.f. [4]. Within the same time bound, this diagram can be preprocessed into a linear size data structure that supports $O(\log |X|)$-time point-location queries, c.f. [6]. This structure can now be queried with each point y of Y to determine which point x of X contains y in its Voronoi cell. Once this is known, the check if $h_P(y)$ is below $\mathcal{C}_P(x)$ can be performed in $O(1)$ time. The total running time of this algorithm is $O((|X| + |Y|) \log |X|)$.

2.2 The Decision Problem

Let C be a polygonal cycle on a set of n vertices in the plane and let $\kappa > 1$ be a real number. In this section, we present an algorithm that decides for each edge e of C, whether or not the dilation of the polygonal path $C \setminus \{e\}$ is less than κ. We first describe the overall approach. Then, we give two implementations that yield running times of $O(n \log^3 n)$ and $O(n \log^2 n)$, respectively.

If $R = (r_1, \ldots r_m)$ and $Q = (q_1, \ldots, q_n)$ are two polygonal paths having the property that $l(R) = r_m = q_1 = f(Q)$, then we denote the concatenation of R and Q by $R \oplus Q$. Thus, $R \oplus Q$ is the polygonal path $(r_1, \ldots, r_m, q_2, \ldots, q_n)$.

In order to facilitate a recursive approach, we will consider the following more general problem: Assume that (the edge set of) C is partitioned into two polygonal paths T (the *top*) and B (the *bottom*) such that $\delta_T < \kappa$. We want to decide for each edge e of B, whether or not the dilation of $C \setminus \{e\}$ is less than κ. If we take $T = \{p\}$, where p is an arbitrary vertex of C, then we obtain the original problem.

The details of the decision algorithm is presented in Algorithm 2.1. The correctness of Algorithm 2.1 is obvious. We will show below that after a preprocessing step taking $O(n \log^2 n)$ expected time, we can decide in $O(|B| \log n)$ time if $\delta_{T \oplus B^r} < \kappa$ and $\delta_{B^l \oplus T} < \kappa$, where $|B|$ denotes the number of edges on B. The expected running time $t(n)$ of the algorithm can therefore be written as

Algorithm 2.1. DECISION-ALGORITHM

Input Paths T and B and $\kappa > 1$.
Output *yes* or *no* for every edge of B.

```
1  if |B| = 1 then
2  |  return yes for the edge in B;
3  else
4  |  l := the first vertex of T;        /* in counterclockwise order along C */
5  |  r := the last vertex of T;         /* in counterclockwise order along C */
6  |  m := the middle vertex of B;
7  |  B^r := the part of the path B between r and m;
8  |  B^l := the part of the path B between m and l;
9  |  if δ_{T⊕B^r} < κ then DECISION-ALGORITHM(T ⊕ B^r, B^l, κ);
10 |  else return no for each edge e of B^l;
11 |  if δ_{B^l⊕T} < κ then DECISION-ALGORITHM(B^l ⊕ T, B^r, κ);
12 |  else return no for each edge e of B^r;
13 end
```

$t(n) = O(n \log^2 n) + r(n)$ where the function r satisfies the recurrence

$$r(b) \leq 2 \cdot r(b/2) + O(b \log n).$$

This implies that $t(n) = O(n \log^2 n)$.

Figure 1 illustrates the recursion tree of Algorithm 2.1. The nodes of the tree are labeled according to a BFS-numbering where the first (left) child of a node corresponds to the recursive call in Step 11 and the second (right) child corresponds to the recursive call in Step 9. Later, we will refer to a recursive call corresponding to the node with BFS-number i as the i-th step of the recursion. For each node i, the current top and bottom paths are denoted by T_i and B_i, respectively. These paths can be computed as follows. Assume that the polygonal cycle C is given by the array $C[0, \ldots, n]$ and that B consists of b vertices. Then $B_1 = C[0, \ldots, b-1]$ and $T_1 = T$. For $i \geq 1$, if $B_i = C[l, r]$, then $B_{2i} := C[l, l + \lfloor \frac{r-l}{2} \rfloor]$, $B_{2i+1} := C[l + \lfloor \frac{r-l}{2} \rfloor + 1, r]$, $T_{2i} := B_{2i+1} \oplus T_i$, and $T_{2i+1} := T_i \oplus B_{2i}$. Observe that each top path T_j is the concatenation of $O(\log n)$ bottom paths.

A First Implementation. We will show that after an $O(n \log^2 n)$–time preprocessing, we can decide if (i) $\delta_{T \oplus B^r} < \kappa$ and (ii) $\delta_{B^l \oplus T} < \kappa$ in $O(|B| \log^2 n)$ time. Later, we will give a faster implementation. Since (ii) is symmetric to (i), we only show how to decide whether or not (i) holds.

Suppose we have a polygonal path T' with $\delta_{T'} < \kappa$ that is given as a list of k polygonal paths $(B'_1, \ldots, B'_k)$ such that $l(B'_i) = f(B'_{i+1})$ for $1 \leq i < k$. Thus, we have $T' = B'_1 \oplus \ldots \oplus B'_k$. Given a new polygonal chain B' with $f(B') = l(T')$, we want to decide if $\delta_{T' \oplus B'} < \kappa$.

Observe that $\delta_{T' \oplus B'} < \kappa$ if and only if (a) $\delta_{T'} < \kappa$, (b) $\delta_{B'} < \kappa$, and (c) $\delta_{T' \oplus B'}(T', B') < \kappa$. We are given that (a) holds. Using the algorithm

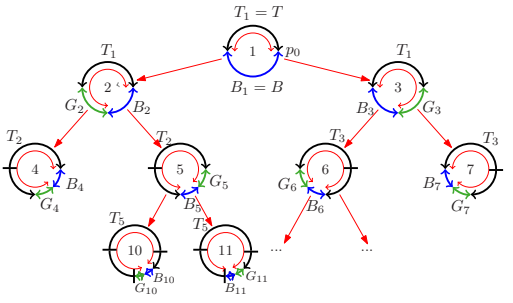


Fig. 1. The recursion tree. Note that $G_{2i} := B_{2i+1}$ and $G_{2i+1} := B_{2i}$.

of [1], we can decide in $O(|B'| \log |B'|)$ time whether or not (b) holds. Thus, it remains to show how to verify whether or not (c) holds.

Obviously, $\delta_{T' \oplus B'}(T', B') < \kappa$ if and only if $\delta_{T' \oplus B'}(B'_i, B') < \kappa$ for each i with $1 \leq i \leq k$. Since $B'_i <_{T' \oplus B'} B'$, we know from Lemma 2 that $\delta_{T' \oplus B'}(B'_i, B') < \kappa$ if and only if $h_{T' \oplus B'}(B')$ lies below $\mathcal{L}(\mathcal{C}_{T' \oplus B'}(B'_i))$.

Assume that for each path B'_i, we have the total accumulated scaled length

$$\ell_i := \sum_{j=1}^{i} d_{B'_j}(l(B'_j), f(B'_j))/\kappa$$

and the additively weighted Voronoi diagram $V_{B'_i}(B'_i)$ that has been augmented with a data structure to support point-location queries in t_{loc} time per query. Recall that $V_{B'_i}(B'_i)$ is the projection of the lower envelope $\mathcal{L}(\mathcal{C}_{B'_i}(B'_i))$ onto the xy-plane. It is defined with respect to the weights $\omega_{B'_i}(p) = d_{B'_i}(p, f(B'_i))/\kappa$. Since $\omega_{T' \oplus B'}(p) = \omega_{B'_i}(p) + \ell_{i-1}$ for all $p \in B'_i$, we have $\omega_{T' \oplus B'}(p) - \omega_{T' \oplus B'}(q) = \omega_{B'_i}(p) - \omega_{B'_i}(q)$ for all $p, q \in B'_i$. It follows that the diagram $V_{B'_i}(B'_i)$ is also the projection of the lower envelope $\mathcal{L}(\mathcal{C}_{T' \oplus B'}(B'_i))$.

The associated point-location structure of B'_i can therefore be used to determine for each point b' in B', the point t in B'_i that contains b' in its Voronoi cell in $V_{T' \oplus B'}(B'_i)$. Once this is known for each point b' in B', we can check if $h_{T' \oplus B'}(b')$ is below $\mathcal{C}_{T' \oplus B'}(t)$. To this end, we compute the weights

$$\omega_{T' \oplus B'}(t) = \omega_{B'_i}(t) + \ell_{i-1} \qquad \text{and} \qquad \omega_{T' \oplus B'}(b') = \omega_{B'}(b') + \ell_k.$$

The overall running time of this approach (excluding the preprocessing time) is $O(k|B'|t_{loc})$.

In our application, the relevant paths B'_i are the bottom paths B_i that appear in the recursive calls. As a consequence, $k = O(\log n)$, $|T' \oplus B'| \leq n$, and we can precompute the required information in $O(n \log^2 n)$ time by computing all the diagrams $V_{B_i}(B_i)$ along with the point-location data structures. Since $t_{loc} = O(\log n)$, it follows that the overall running time of this approach (after $O(n \log^2 n)$ preprocessing time) is $O(|B'| \log^2 n)$. With this implementation, Algorithm 2.1 runs in $O(n \log^3 n)$ time.

A Faster Implementation. In the i-th step of the recursion, we split B_i (almost evenly) into B_{2i} and B_{2i+1}, and compute the diagrams $V_{B_{2i}}(B_{2i})$ and $V_{B_{2i+1}}(B_{2i+1})$. We then locate each point b of B_{2i} in $V_{B_{2i+1}}(B_{2i+1})$ to determine which point t of B_{2i+1} contains b in its Voronoi cell in $V_{B_{2i+1}}(B_{2i+1})$. We store t in a table $\mathcal{T}_b$ associated with b under the key $2i+1$ that identifies the set B_{2i+1}. In the same way, we locate each point b of B_{2i+1} in $V_{B_{2i}}(B_{2i})$ and store the corresponding point t of B_{2i} in a table $\mathcal{T}_b$ associated with b under the key $2i$.

Since we perform exactly one point-location query for each point b of B_1 on each level of the recursion tree, the table $\mathcal{T}_b$ has $O(\log n)$ entries. We can therefore use the construction of [5] to store $\mathcal{T}_b$ in a perfect-hash table of size $O(\log n)$ that supports $O(1)$ access time. Note that in the complexity model of [5], they assume that the entries come from a universe set and the memory is able to randomly access each entry in constant time. Recall that the construction of [5] is randomized and builds the hash table in $O(\log n)$ expected time.

The total time we spend on each level of the recursion tree is $O(n \log n)$, so the total expected preprocessing time is $O(n \log^2 n)$ and the total time we spend for answering point-location queries is $O(n \log^2 n)$.

In order to determine for a point b' of B', where $B' \subseteq B_1$, which point t of B'_i contains b' in its Voronoi cell, we find the index j for which $B'_i = B_j$. Then we retrieve the entry with the key j from $\mathcal{T}_{b'}$. This is exactly the point t of B_j that contains b in its Voronoi cell in $V_{B_j}(B_j)$.

It follows that $t_{loc} = O(1)$, so that the overall running time of this approach (after $O(n \log^2 n)$ preprocessing time) is $O(|B'| \log n)$. With this implementation, Algorithm 2.1 runs in $O(n \log^2 n)$ expected time.

2.3 The Optimization Algorithm

We now present our algorithm that computes, for a given polygonal cycle C on a set of n points in the plane, the value of $\delta_C^{\min}$ in $O(n \log^3 n)$ expected time. Clarkson and Shor [3] used a similar randomized approach to compute diameter of a point set.

Step 1: Compute a random permutation of the edges of C. We denote the permutation by $e_1, e_2, \ldots, e_n$.
Step 2: Use the algorithm of [1] to compute the dilation of the path $C \setminus \{e_1\}$ and assign this value to κ.
Step 3: Run Algorithm 2.1 and store with each edge e of C a Boolean flag which is *true* if and only if the dilation of the path $C \setminus \{e\}$ is less than κ.
Step 4: For $i = 2, 3, \ldots, n$, do the following: If the flag stored with e_i is *true*, then perform the following Steps 4.1 and 4.2:
Step 4.1: Use the algorithm of [1] to compute the dilation of the path $C \setminus \{e_i\}$ and assign this value to κ.
Step 4.2: Run Algorithm 2.1 and store with each edge e of C a Boolean flag which is *true* if and only if the dilation of the path $C \setminus \{e\}$ is less than κ.
Step 5: Return the value of κ.

The correctness of the algorithm follows from the fact that, after Step 4, $\kappa = \min_{1 \le i \le n} \delta_{P_i} = \delta_C^{\min}$, where P_i is the polygonal path obtained by removing e_i from C.

Clearly, Step 1 takes $O(n)$ time. The algorithm of [1] and, therefore, Step 2, takes $O(n \log n)$ expected time. Each time we run Algorithm 2.1, we spend $O(n \log^2 n)$ expected time. Observe that we run this algorithm once in Step 3 and, moreover, in Step 4 each time the dilation of P_i is less than the current value of κ. In the latter case, we also spend $O(n \log n)$ expected time to compute the dilation of P_i. Since the edges of C are in random order, the values $\delta_{P_1}, \ldots, \delta_{P_n}$ are in random order as well. At the start of the i-th iteration of Step 4, the value of κ is equal to $\min_{1 \le j < i} \delta_{P_j}$. Thus, $\delta_{P_i} < \kappa$ if and only if δ_{P_i} is the minimum of the set $\{\delta_{P_j} \mid 1 \le j \le i\}$. It follows that $\delta_{P_i} < \kappa$ with probability $1/i$. Using the linearity of expectation, it follows that the expected number of times that Steps 4.1 and 4.2 are performed is equal to $\sum_{i=2}^{n} 1/i = O(\log n)$. Thus, the overall expected running time of our algorithm is $O(n \log^3 n)$. This completes the proof of Theorem 1.

3 Dilation-Maximal Edge Deletion in a Polygonal Cycle

In this section, we will prove Theorem 2. That is, we give an algorithm that computes $\delta_C^{\max} = \max_{0 \le i < n} \delta_{P_i}$.

Let L be the total length of the edges of C. We define $\Delta(p_0) := 0$ and $\Delta(p_i) := \Delta(p_{i-1}) + |p_{i-1}p_i|$ for $1 \le i < n$. Thus, $\Delta(p_i)$ is the length of the path $(p_0, \ldots, p_i)$ and the shortest-path distance $d_C(p_i, p_j)$ between p_i and p_j in the cycle C is given by $d_C(p_i, p_j) = \min(|\Delta(p_i) - \Delta(p_j)|, L - |\Delta(p_i) - \Delta(p_j)|)$.

Consider two distinct vertices x and y of C. We obtain the largest dilation between x and y in any path P_i, by deleting an arbitrary edge on the shorter of the two paths in C between x and y. Thus, the following lemma holds.

Lemma 3. *Let x and y be two distinct vertices of C. Then*

$$\max_{0 \le i < n} \delta_{P_i}(x, y) = \frac{\max(|\Delta(x) - \Delta(y)|, L - |\Delta(x) - \Delta(y)|)}{|xy|} \ge \frac{L}{2|xy|}.$$

The next lemma states that the closest pair in the vertex set of C can be used to obtain a 2-approximation to $\delta_C^{\max}$.

Lemma 4. *Let (p, q) be a closest pair in the vertex set of C. Then*

$$\delta_C^{\max} \le 2 \cdot \max_{0 \le i < n} \delta_{P_i}(p, q).$$

Proof. Let j be an index such that $\delta_C^{\max} = \delta_{P_j}$ and let x and y be two vertices of C such that $\delta_{P_j} = \delta_{P_j}(x, y)$. Then $\delta_C^{\max} = \frac{d_{P_j}(x,y)}{|xy|} \le \frac{L}{|pq|}$. By Lemma 3, we have $\frac{L}{|pq|} \le 2 \cdot \max_{0 \le i < n} \delta_{P_i}(p, q)$. □

Thus, by computing the closest pair (p, q) in the vertex set of C and then using Lemma 3 to compute $\max_{0 \le i < n} \delta_{P_i}(p, q)$, we obtain a 2-approximation to $\delta_C^{\max}$.

We now show that a simple extension leads to an algorithm that computes the exact value of $\delta_C^{\max}$.

Let S be the set of all pairs (x, y) in the vertex set of C for which $x \neq y$ and $|xy| \leq 2|pq|$. The following lemma states that it suffices to consider the elements of S to compute $\delta_C^{\max}$.

Lemma 5. *We have* $\delta_C^{\max} = \max_{(x,y)\in S} \max_{0\leq i<n} \delta_{P_i}(x, y)$.

Proof. It is clear that

$$\delta_C^{\max} = \max_{x,\, y \text{ vertices of } C} \max_{0\leq i<n} \delta_{P_i}(x, y) \geq \max_{(x,y)\in S} \max_{0\leq i<n} \delta_{P_i}(x, y).$$

Let j be an index such that $\delta_C^{\max} = \delta_{P_j}$ and let a and b be two vertices of C such that $\delta_{P_j} = \delta_{P_j}(a, b)$. If we can show that $(a, b) \in S$ (i.e., $|ab| \leq 2|pq|$), then the proof is complete.

By Lemma 3, we have the following inequality, which follows that

$$\begin{aligned}
\frac{L}{2|pq|} &\leq \max_{0\leq i<n} \delta_{P_i}(p, q) \\
&\leq \max_{x,\, y \text{ vertices of } C} \max_{0\leq i<n} \delta_{P_i}(x, y) \\
&= \delta_C^{\max} \\
&= \delta_{P_j}(a, b) \\
&= d_{P_j}(a, b)/|ab| \\
&\leq L/|ab|.
\end{aligned}$$

This implies that $|ab| \leq 2|pq|$. □

The discussion above leads to the following algorithm for computing the value of $\delta_C^{\max}$.

Step 1: Compute the total length L of the cycle C and compute the values $\Delta(p_i)$ $(0 \leq i < n)$ as defined above.
Step 2: Compute the closest pair (p, q) in the vertex set of C.
Step 3: Compute the set S of all pairs (x, y) in the vertex set of C for which $x \neq y$ and $|xy| \leq 2|pq|$.
Step 4: For each element (x, y) in S, compute

$$\frac{\max(|\Delta(x) - \Delta(y)|, L - |\Delta(x) - \Delta(y)|)}{|xy|}.$$

Step 5: Return the largest value computed in Step 4.

By Lemma 3, each value computed in Step 4 is equal to $\max_{0\leq i<n} \delta_{P_i}(x, y)$. By Lemma 5, the largest of the values computed in Step 4 is equal to $\delta_C^{\max}$. This proves the correctness of the algorithm. To analyze the running time of the algorithm, it is clear that Step 1 takes $O(n)$ time. The closest-pair computation in Step 2 takes $O(n \log n)$ time; see [12]. In [9], it is shown that the size of the

set S is $O(n)$. It is also shown there that if the points in the vertex set of C are stored in two lists X and Y, where the points in X are sorted by x-coordinates and the points in Y are sorted by y-coordinates, and if there are cross-pointers between these two lists, then the set S can be computed in $O(n)$ time. Therefore, Step 3 takes $O(n \log n)$ time. In Step 4, the algorithm spends $O(1)$ time for each element of S. Since the size of S is $O(n)$, the total time for Step 4 is $O(n)$. Thus, the total time of the algorithm is $O(n \log n)$.

If the cycle C is a convex polygon, then we can improve the running time: In [8], it is shown that the closest pair can be computed in $O(n)$ time. Since C is a convex polygon, we can obtain the lists X and Y in $O(n)$ time. It follows that the entire algorithm runs in $O(n)$ time.

4 Approximating the Dilation of All Paths P_i

Consider again the polygonal cycle $C = (p_0, \ldots, p_{n-1}, p_0)$ whose vertices are points in the plane. Let $\epsilon > 0$ be a constant. In this section, we prove Theorem 3. That is, we show that an approximation to the dilation of *each* path P_i $(0 \leq i < n)$, as well as an approximation to $\delta_C^{\min}$, can be computed in $O(n \log n)$ total time.

Our algorithm uses the *well-separated pair decomposition (WSPD)* of [2]. More specifically, we use the following lemma from [10], which states that the WSPD of the vertex set of any Euclidean graph G can be used to approximate the dilation of G. The statement of the lemma as we present it appears in Section 13.2.1 of [11].

Lemma 6. *Let V be a set of n points in the plane and let $\{A_1, B_1\}, \{A_2, B_2\}, \ldots, \{A_m, B_m\}$ be a WSPD for V with separation ratio $4(2+\epsilon)/\epsilon$. For each j with $1 \leq j \leq m$, let a_j be an arbitrary point in A_j, and let b_j be an arbitrary point in B_j. For any connected Euclidean graph G with vertex set V, the following holds: For each j with $1 \leq j \leq m$, let $\delta_G(a_j, b_j)$ be the dilation between a_j and b_j in G, and let*

$$t := \max_{1 \leq j \leq m} \delta_G(a_j, b_j).$$

Then $\delta_G/(1+\epsilon) \leq t \leq \delta_G$, where δ_G denotes the dilation of G.

Thus, in order to approximate the dilation of a Euclidean graph, it is sufficient to compute the dilation between $O(n)$ pairs of vertices. Moreover, the choice of these vertices depends only on the vertex set of the graph, it does not depend on the edges of the graph.

In a preprocessing step, we use the algorithm of [2] to compute, in $O(n \log n)$ time, a WSPD $\{A_j, B_j\}$, $1 \leq j \leq m = O(n)$, for the vertex set $\{p_0, \ldots, p_{n-1}\}$ of the cycle C, with separation ratio $4(2+\epsilon)/\epsilon$. For each j with $1 \leq j \leq m$, we pick an arbitrary point a_j in A_j, and an arbitrary point b_j in B_j. Our algorithm will compute, for each i with $0 \leq i < n$, the value $t_i := \max_{1 \leq j \leq m} \delta_{P_i}(a_j, b_j)$. Observe that, by Lemma 6, $\delta_{P_i}/(1+\epsilon) \leq t_i \leq \delta_{P_i}$.

Lemma 7. *Let $t^* := \min(t_0, t_1, \ldots, t_{n-1})$. Then $\delta_C^{\min}/(1+\epsilon) \leq t^* \leq \delta_C^{\min}$.*

Proof. Let i be an index such that $t^* = t_i$ and let j be an index such that $\delta_C^{\min} = \delta_{P_j}$. Then $t^* \le t_j \le \delta_{P_j} = \delta_C^{\min}$ and $\delta_C^{\min} = \delta_{P_j} \le \delta_{P_i} \le (1+\epsilon)t_i = (1+\epsilon)t^*$. □

We remark that, by a similar argument, $t^{**} := \max(t_0, t_1, \ldots, t_{n-1})$ can be shown to satisfy $\delta_C^{\max}/(1+\epsilon) \le t^{**} \le \delta_C^{\max}$. In other words, the algorithm that will be presented below can also be used to compute an approximation to $\delta_C^{\max}$ in $O(n \log n)$ time. We have seen in Sect. 3, however, that the exact value of $\delta_C^{\max}$ can be computed within the same time bound.

As mentioned above, our algorithm computes t_i for $i = 0, 1, \ldots, n-1$. The main idea is to maintain, for the current value of i, the m dilations $\delta_{P_i}(a_j, b_j)$ $(1 \le j \le m)$ in a balanced binary search tree T. Observe that, for any fixed index j, the value of $\delta_{P_i}(a_j, b_j)$ changes at most twice when i is increased from 0 to $n-1$. As a result, the total number of updates in T will be at most $2m$. We now present the details.

Let P denote the path $(p_0, p_1, \ldots, p_{n-1})$. Recall the relation $<_P$ of Sect. 2.1. We may assume without loss of generality that $a_j <_P b_j$ for each j with $1 \le j \le m$.

In the preprocessing step, we compute, in $O(n)$ time, the values $\Delta(p_i) = d_P(p_0, p_i)$ $(0 \le i < n)$ and the total length L of the cycle C. Observe that, for $0 \le i < n$, the distance $d_{P_i}(a_j, b_j)$ between a_j and b_j in the path P_i satisfies

$$d_{P_i}(a_j, b_j) = \begin{cases} \Delta(b_j) - \Delta(a_j) & \text{if } i = n-1 \text{ or } p_i <_P a_j \text{ or } b_j <_P p_{i+1}, \\ L - (\Delta(b_j) - \Delta(a_j)) & \text{otherwise.} \end{cases} \tag{1}$$

In the final part of the preprocessing step, we compute, for each i with $0 < i < n$, the set $S_i := \{j \mid 1 \le j \le m, a_j = p_i \text{ or } b_j = p_i\}$. Obviously, all these sets can be computed in $O(m) = O(n)$ time. After the preprocessing step, the algorithm proceeds as follows.

Step 1: Initialize an empty balanced binary search tree T (e.g., a red-black tree).
Step 2: For $j = 1, 2, \ldots, m$, use (1) to compute $d_{P_0}(a_j, b_j)$, compute $D_j := \delta_{P_0}(a_j, b_j)$ and insert it into T.
Step 3: Compute the maximum element t_0 in the tree T.
Step 4: For $i = 1, 2, \ldots, n-1$, perform the following Steps 4.1–4.2. (Observe that, at this moment, $D_j = \delta_{P_{i-1}}(a_j, b_j)$, $1 \le j \le m$.)
  Step 4.1: For each element j in S_i, delete D_j from the tree T, use (1) to compute $d_{P_i}(a_j, b_j)$, compute the new value $D_j := \delta_{P_i}(a_j, b_j)$ and insert it into T.
  Step 4.2: Compute the maximum element t_i in the tree T.
Step 5: Compute $t^* := \min_{0 \le i < n} t_i$, and return the sequence $t_0, t_1, \ldots, t_{n-1}, t^*$.

The correctness of the algorithm follows from the discussion above. We have seen already that the preprocessing step takes $O(n \log n)$ time. Steps 1–3 take $O(m \log m) = O(n \log n)$ time. The total time for Step 4 is proportional to

$$\sum_{i=1}^{n-1} (|S_i| + 1) \log m \le (2m + n) \log m = O(n \log n).$$

This completes the proof of Theorem 3.

5 Concluding Remarks and Acknowledgments

Recently, there has been a fair amount of work on the problem of computing the optimal dilation of a given (geometric) graph. In this paper we considered a variation of the problem where we are given a polygonal cycle and are supposed to choose one edge to remove such that the resulting polygonal path gives the smallest (or the largest) possible dilation.

We would like to thank Jan Vahrenhold for fruitful discussions on the subject.

References

[1] Agarwal, P.K., Klein, R., Knauer, C., Sharir, M.: Computing the detour of polygonal curves. Technical Report B 02-03, Fachbereich Mathematik und Informatik, Freie Universität Berlin (2002)

[2] Callahan, P.B., Kosaraju, S.R.: A decomposition of multidimensional point sets with applications to k-nearest-neighbors and n-body potential fields. Journal of the ACM 42, 67–90 (1995)

[3] Clarkson, K.L., Shor, P.W.: Algorithms for diametral pairs and convex hulls that are optimal, randomized, and incremental. In: SCG 1988: Proceedings of the fourth annual symposium on Computational geometry, pp. 12–17. ACM Press, New York (1988)

[4] Fortune, S.J.: A sweepline algorithm for Voronoi diagrams. Algorithmica 2, 153–174 (1987)

[5] Fredman, M.L., Komlos, J., Szemeredi, E.: Storing a sparse table with $O(1)$ worst case access time. Journal of the ACM 31, 538–544 (1984)

[6] Kirkpatrick, D.G.: Optimal search in planar subdivisions. SIAM Journal on Computing 12, 28–35 (1983)

[7] Langerman, S., Morin, P., Soss, M.: Computing the maximum detour and spanning ratio of planar paths, trees and cycles. In: Alt, H., Ferreira, A. (eds.) STACS 2002. LNCS, vol. 2285, pp. 250–261. Springer, Heidelberg (2002)

[8] Lee, D.T., Preparata, F.P.: The all nearest-neighbor problem for convex polygons. Information Processing Letters 7, 189–192 (1978)

[9] Lenhof, H.P., Smid, M.: Sequential and parallel algorithms for the k closest pairs problem. International Journal of Computational Geometry & Applications 5, 273–288 (1995)

[10] Narasimhan, G., Smid, M.: Approximating the stretch factor of Euclidean graphs. SIAM Journal on Computing 30, 978–989 (2000)

[11] Narasimhan, G., Smid, M.: Geometric Spanner Networks. Cambridge University Press, Cambridge (2007)

[12] Smid, M.: Closest-point problems in computational geometry. In: Sack, J.-R., Urrutia, J. (eds.) Handbook of Computational Geometry, pp. 877–935. Elsevier Science, Amsterdam (2000)

Unbounded-Error Classical and Quantum Communication Complexity

Kazuo Iwama[1,⋆], Harumichi Nishimura[2,⋆⋆], Rudy Raymond[3], and Shigeru Yamashita[4,⋆⋆⋆]

[1] School of Informatics, Kyoto University, Kyoto 606-8501, Japan
iwama@kuis.kyoto-u.ac.jp
[2] School of Science, Osaka Prefecture University, Sakai 599-8531, Japan
hnishimura@mi.s.osakafu-u.ac.jp
[3] Tokyo Research Laboratory, IBM Japan, Yamato 242-8502, Japan
raymond@jp.ibm.com
[4] Graduate School of Information Science, Nara Institute of Science and Technology Ikoma 630-0192, Japan
ger@is.naist.ac.jp

Abstract. Since the seminal work of Paturi and Simon [26, FOCS'84 & JCSS'86], the unbounded-error classical communication complexity of a Boolean function has been studied based on the arrangement of points and hyperplanes. Recently, [14, ICALP'07] found that the unbounded-error *quantum* communication complexity in the *one-way communication* model can also be investigated using the arrangement, and showed that it is exactly (without a difference of even one qubit) half of the classical one-way communication complexity. In this paper, we extend the arrangement argument to the *two-way* and *simultaneous message passing* (SMP) models. As a result, we show similarly tight bounds of the unbounded-error two-way/one-way/SMP quantum/classical communication complexities for *any* partial/total Boolean function, implying that all of them are equivalent up to a multiplicative constant of four. Moreover, the arrangement argument is also used to show that the gap between *weakly* unbounded-error quantum and classical communication complexities is at most a factor of three.

1 Introduction

As with many other probabilistic computation models, *communication complexity* (CC for short) has two contradistinctive settings: *Bounded-error* CC refers to the amount of communication (the number of bits exchanged) between Alice and Bob which is enough to compute a Boolean value $f(x, y)$, with high probability, from Alice's input x and Bob's input y. On the other hand, *unbounded-error* CC refers to the lowest possible amount of communication which is needed to give

⋆ Research supported in part by KAKENHI (16092101, 19200001).
⋆⋆ Research supported in part by KAKENHI 19700011.
⋆⋆⋆ Research supported in part by KAKENHI (16092218,19700010).

T. Tokuyama (Ed.): ISAAC 2007, LNCS 4835, pp. 100–111, 2007.

"a positive hint" for the computation of $f(x, y)$, in other words, even one-bit less communication would be the same as completely no communication in the worst case. More formally, it is defined as the minimum amount of communication between Alice and Bob such that for all x and y Alice (or Bob) can output a correct value of $f(x, y)$ with probability $> 1/2$.

Unbounded-error CC was first studied by Paturi and Simon [26], who characterized its one-way version, $C^1(f)$, in terms of the minimum dimension k_f of the *arrangement* that realizes the Boolean function f (see Sec. 2 for the definition of arrangements). Namely they showed $\lceil \log k_f \rceil \leq C^1(f) \leq \lceil \log(k_f + 2) \rceil$. It was also proven that the two-way (unbounded-error) CC, $C(f)$, does not differ from $C^1(f)$ more than one bit for any (partial or total) Boolean function f, which is a bit surprising since there are easily seen exponential differences between them in the bounded-error setting (see, say [21]).

Since then, arrangement has been a standard tool for studying unbounded-error CC. Alon, Frankl, and Rödl [1] showed by counting arguments that almost all Boolean functions have linear unbounded-error CCs. The first linear lower bound of an explicit function was found by Forster [8], who gave the linear lower bound of the inner product function by showing the lower bound of its minimum dimension using operator norms. Extending Forster's arguments, there are several papers on the study of unbounded-error CC [9,10] that also put emphasis on the margin of arrangements.

Recently, [14] completely characterized the unbounded-error one-way (Alice to Bob) quantum CC, $Q^1(f)$, also in terms of k_f, i.e., $Q^1(f) = \lceil \log \sqrt{k_f + 1} \rceil$. The main idea was to relate quantum states in Alice's side and POVMs in Bob's side to points and hyperplanes of a real space arrangement, respectively. Moreover, they also closed the small gap between the upper and lower bounds of $C^1(f)$ in [26] by proving $C^1(f) = \lceil \log(k_f + 1) \rceil$. As a result, they found that the unbounded-error one-way quantum CC of any Boolean function is always exactly one half of its classical counterpart. Unfortunately, however, their studies were limited within the one-way model: The proof technique mentioned above apparently depends on the one-way communication and there is no obvious way of its extension to the more general two-way communication model. Furthermore, it seems hard to change two-way *quantum* protocols to one-way quantum protocols efficiently, which was possible and was used as the basic approach in the classical case [26].

Our Contribution. We provide a new approach for constructing an arrangement from a given two-way quantum protocol with n qubit communication. The basic idea is to use the simple fact, found by Yao [30] and Kremer [20], that the final state of the whole system after the protocol is finished can be written as a superposition of at most 2^n different states. This allows us to imply a quite tight lower bound for the two-way quantum CC $Q(f)$, namely $Q(f) \geq \lceil \log \sqrt{k_f + 1/8} - 1/2 \rceil$. Notice that this lower bound does not differ more than one qubit from the upper bound of one-way CC $Q^1(f)$ in [14], which then means that all of $Q(f)$, $Q^1(f)$, $C(f)/2$ and $C^1(f)/2$ coincide within the difference of at most only one bit or one qubit.

Arrangements are also useful to provide a couple of related results: First, we give almost tight characterizations of $Q^{||}(f)$ and $C^{||}(f)$, i.e., the unbounded-error quantum and classical CCs in the *simultaneous message passing* (SMP) model. We prove that $Q^{||}(f)$ and $C^{||}(f)$ are equal to twice as much as $Q^1(f)$ and $C^1(f)$ up to a few qubits and bits, respectively. Therefore we can see that in the unbounded-error setting all of the two-way/one-way/SMP quantum/classical CCs of any Boolean function are asymptotically equivalent up to a multiplicative constant of four. Note that, in the bounded-error classical case, the equality function gives an exponential gap between one-way and SMP CCs [4,24]. In the bounded-error quantum case, it is also shown that an exponential gap between one-way and SMP CCs exists for some *relations* [12].

Secondly, we give several relations among CCs in the *weakly unbounded-error* setting, which was introduced by Babai et al. [3]. The weakly unbounded-error (classical) CC of a protocol P, denoted by $C_w(P)$, is measured by the sum of the communication cost of P and $\log 1/(p - 1/2)$ if P's success probability is p. The weakly unbounded-error CC of f, $C_w(f)$, is the minimum of $C_w(P)$ over all protocols P that computes f. The quantum variant and one-way/SMP variants are defined similarly. Using two quantities of arrangement, margin and dimension, we show several upper bounds of weakly unbounded-error CCs, in particular, $C_w(f) \leq 3Q_w(f) + O(1)$. Previously, it is only known [17] that $C_w(f) = O(Q_w(f))$. The multiplicative factor three seems to be quite tight since at least a factor of two must be involved as a gap between quantum and classical communication costs as mentioned before.

Related Work. In the bounded-error setting, CCs of some Boolean functions have large gaps between quantum and classical cases: Exponential separations are known for all of two-way [27], one-way [11] and SMP models [5], where the first two cases are for partial Boolean functions, and the last case is for a total Boolean function. It remains to show (if any) exponential gaps for total Boolean functions in the cases of two-way and one-way models. In particular, the largest known gap between quantum and classical one-way CCs is only a factor of two.

Other than the minimum dimension k_f of arrangements, several different measures of Boolean functions also appeared in the literature. Paturi and Simon [26] showed that $C^1(f)$ (and $C(f)$) is equal to the logarithm of the *sign-rank*, $srank(f)$, up to a few bits (also see [6]). Due to Klauck [17], both $C_w(f)$ and $Q_w(f)$ are equivalent to the logarithm of the inverse of the *discrepancy* $disc(f)$ (see, say [21]) within a constant multiplicative factor and a logarithmic additive factor. The recent result by Linial and Shraibman [22,23] implies that the maximal margin of arrangements realizing f, $m(f)$, is equivalent to $disc(f)$ up to a multiplicative constant. Thus, combined with the results of the current paper, (i) $C(f)$, $Q(f)$, $\log k_f$ and $\log srank(f)$ are all within a factor of two, and (ii) $C_w(f)$, $Q_w(f)$, $\log disc^{-1}(f)$ and $\log m^{-1}(f)$ within a factor of some constant and a logarithmic additive term. However, due to the two independent results by Buhrman et al. [6] and Sherstov [28], (i) is exponentially smaller than (ii) for some Boolean function f.

2 Technical Components

In this section, we present some basic tools for obtaining our results. Their proofs, as well as some of those in the following sections, are omitted due to space constraints. They are mainly the concept of arrangement and its sufficient conditions (Lemmas 3 and 4) for realizing a quantum protocol whose success probability can be calculated from arrangement parameters by Lemma 5 in [14].

Arrangements. We denote a point in $\mathbb{R}^n$ by the corresponding n-dimensional real vector, and a hyperplane $\{(a_i) \in \mathbb{R}^n \mid \sum_{i=1}^n a_i h_i = h_{n+1}\}$ by the $(n+1)$-dimensional real vector $\mathbf{h} = (h_1, \ldots, h_n, h_{n+1})$, meaning that any point (a_i) on the plane satisfies the equation $\sum_{i=1}^n a_i h_i = h_{n+1}$. A Boolean function f on $X \times Y$ is *realizable by an arrangement* of a set of $|X|$ points $\mathbf{p}_x = (p_1^x, \ldots, p_k^x)$ and a set of $|Y|$ hyperplanes $\mathbf{h}_y = (h_1^y, \ldots, h_k^y, h_{k+1}^y)$ in $\mathbb{R}^k$ if for any $x \in X$ and $y \in Y$, $\mathrm{sign}(\sum_{i=1}^k p_i^x h_i^y - h_{k+1}^y) = f(x,y)$. Here, $\mathrm{sign}(a) = 1$ if $a > 0$ and -1 if $a < 0$. The value $\left|\sum_{i=1}^k p_i^x h_i^y - h_{k+1}^y\right|$ denotes how far the point $\mathbf{p}_x$ lies from the plane $\mathbf{h}_y$, and the *margin* of an arrangement denotes the smallest of such values in the arrangement. The *magnitude* of the arrangement is defined as $\max_{x,y}\left(\sqrt{\sum_{i=1}^k |p_i^x|^2}, \sqrt{\sum_{i=1}^k |h_i^y|^2}, |h_{k+1}^y|\right)$. The value k is called the *dimension* of the arrangement. Let k_f denote the minimum dimension of all arrangements that realize f.

Remark. In the hereafter, our statements will use "functions" while their proofs, that obviously hold for partial, are showed only for total ones. Note also that the concept of arrangement in this paper is not *symmetric*. Here, Alice's input x and Bob's input y are associated with a point and a hyperplane, respectively. For this reason, the value of k_f might be different from that of k_{f^t}, where $f^t(x,y) := f(y,x)$. However, it can be easily seen that $|k_f - k_{f^t}| \leq 1$. The random access coding is one of examples such that $|k_f - k_{f^t}| = 1$ [2,14].

The following lemma relates arrangements to classical CC, which was shown in [26] and later in [9] in more detail including the margin.

Lemma 1 (From arrangements to classical CC). *Any N-dimensional arrangement realizing f of magnitude at most 1 with margin μ can be converted into a classical one-way protocol for f using at most $\lceil \log(N+1) \rceil + 1$ bits with success probability at least $1/2 + \mu/(2\sqrt{N+1})$.*

Bloch Vector Representations of Quantum States. Let $N = 2^n$. Any n-qubit state can be represented by an $N \times N$ positive matrix $\boldsymbol{\rho}$ (also often called N-level quantum state), satisfying $\mathrm{Tr}(\boldsymbol{\rho}) = 1$. Moreover, $\boldsymbol{\rho}$ can be written as a linear combination of N^2 *generator matrices* $\boldsymbol{I}_N, \boldsymbol{\lambda}_1, \ldots, \boldsymbol{\lambda}_{N^2-1}$, where $\boldsymbol{I}_N$ is the identity matrix (the subscript N is often omitted when it is clear from the context), and $\boldsymbol{\lambda}_i$'s are $N \times N$ matrices which are generators of $SU(N)$ satisfying (i) $\boldsymbol{\lambda}_i = \boldsymbol{\lambda}_i^\dagger$ (i.e., $\boldsymbol{\lambda}_i$'s are Hermitian), (ii) $\mathrm{Tr}(\boldsymbol{\lambda}_i) = 0$ and (iii) $\mathrm{Tr}(\boldsymbol{\lambda}_i \boldsymbol{\lambda}_j) = 2\delta_{ij}$. Note that $\boldsymbol{\lambda}_i$ can be any generator matrices satisfying the above conditions (and in

fact N can be any positive integer ≥ 2), but practically for $n = 1$ one can choose $\boldsymbol{\sigma}_1 = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$, $\boldsymbol{\sigma}_2 = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$, and $\boldsymbol{\sigma}_3 = \begin{pmatrix} 0 & -\imath \\ \imath & 0 \end{pmatrix}$ of Pauli matrices as $\boldsymbol{\lambda}_1, \boldsymbol{\lambda}_2$, and $\boldsymbol{\lambda}_3$, respectively. For larger n, one can choose the following tensor products of Pauli matrices for $\boldsymbol{\lambda}_1, \ldots, \boldsymbol{\lambda}_{N^2-1}$: $\boldsymbol{\lambda}_1 = \sqrt{\frac{2}{N}}\boldsymbol{I}_2^{\otimes n-1} \otimes \boldsymbol{\sigma}_1$, $\boldsymbol{\lambda}_2 = \sqrt{\frac{2}{N}}\boldsymbol{I}_2^{\otimes n-1} \otimes \boldsymbol{\sigma}_2$, $\boldsymbol{\lambda}_3 = \sqrt{\frac{2}{N}}\boldsymbol{I}_2^{\otimes n-1} \otimes \boldsymbol{\sigma}_3, \boldsymbol{\lambda}_4 = \sqrt{\frac{2}{N}}\boldsymbol{I}_2^{\otimes n-2} \otimes \boldsymbol{\sigma}_1 \otimes \boldsymbol{I}, \ldots, \boldsymbol{\lambda}_{N^2-2} = \sqrt{\frac{2}{N}}\boldsymbol{\sigma}_3^{\otimes n-1} \otimes \boldsymbol{\sigma}_2$, and $\boldsymbol{\lambda}_{N^2-1} = \sqrt{\frac{2}{N}}\boldsymbol{\sigma}_3^{\otimes n}$. The following representation is known on N-level quantum states (see, e.g., [16]).

Lemma 2. *For any N-level quantum state $\boldsymbol{\rho}$ and any $N \times N$ generator matrices $\boldsymbol{\lambda}_i$'s, there exists an (N^2-1)-dimensional vector $\mathbf{r} = (r_i)$ such that $\boldsymbol{\rho}$ can be written as*

$$\boldsymbol{\rho} = \frac{1}{N}\left(\boldsymbol{I} + \sqrt{\frac{N(N-1)}{2}}\sum_{i=1}^{N^2-1} r_i \boldsymbol{\lambda}_i\right).$$

The vector $\mathbf{r}$ in this lemma is often called the *Bloch vector* of $\boldsymbol{\rho}$.

Note that Lemma 2 is a necessary condition for $\boldsymbol{\rho}$ to be a quantum state. The following sufficient condition appeared in [14], using the geometric fact of Bloch vectors in [15,19].

Lemma 3. *For any $\mathbf{r} = (r_1, r_2, \ldots, r_k) \in \mathbb{R}^k$ and any N satisfying $N^2 \geq k+1$,*

$$\boldsymbol{\rho}(\mathbf{r}) = \frac{1}{N}\left(\boldsymbol{I} + \sqrt{\frac{N(N-1)}{2}}\sum_{i=1}^{k}\left(\frac{r_i}{|\mathbf{r}|(N-1)}\right)\boldsymbol{\lambda}_i\right)$$

is an N-level quantum state. (Intuitively, if a vector is shrunk enough to be inside the ball of radius $1/(N-1)$, its shrunk vector is always a quantum state.) Moreover, if $\boldsymbol{\rho}(\mathbf{r})$ is a quantum state, then $\boldsymbol{\rho}(\gamma\mathbf{r})$ is also a quantum state for any $0 \leq \gamma \leq 1$.

Bloch Vector Representations of POVMs. A POVM $M = \{\boldsymbol{E}, \boldsymbol{I} - \boldsymbol{E}\}$ is a set of operators, which represents a quantum measurement, such that $\boldsymbol{E}$ and $\boldsymbol{I} - \boldsymbol{E}$ are positive matrices. It is known that any POVM M on N-level quantum states can be written as a linear combination of $N \times N$ generator matrices $\boldsymbol{\lambda}_i$'s. Namely,

$$\boldsymbol{E} = e_{N^2}\boldsymbol{I} + \sum_{i=1}^{N^2-1} e_i \boldsymbol{\lambda}_i,$$

where $\mathbf{e} = (e_1, e_2, \ldots, e_{N^2})$ is called the *Bloch vector representation* of POVM M. One sufficient condition for a vector to represent a POVM is given as follows.

Lemma 4. *Let $\mathbf{e} = (e_1, e_2, \ldots, e_{N^2}) \in \mathbb{R}^{N^2}$ such that*

$$\sum_{i=1}^{N^2-1} e_i^2 \leq \frac{N}{2(N-1)}\min(e_{N^2}^2, (1-e_{N^2})^2).$$

If we take $\boldsymbol{E} = e_{N^2}\boldsymbol{I} + \sum_{i=1}^{N^2-1} e_i\boldsymbol{\lambda}_i$, *then* $\{\boldsymbol{E}, \boldsymbol{I} - \boldsymbol{E}\}$ *is a POVM on* N*-level quantum states.*

3 Two-Way Communication Complexity

The model is due to Yao [30]: The space of a quantum protocol consists of Alice and Bob's private parts and a communication channel. On her (his) turn, Alice (Bob) applies a unitary transformation on her (his) part and the communication channel, and Bob (Alice) receives quantum information from the content of the channel. Finally the output of the protocol is obtained by a measurement via Alice or Bob. Note that without loss of generality we can assume that no measurement is performed in the middle of the protocol. This is because it is well known that measurements can be postponed without increasing the communication cost [25]. Also, it is often assumed, for technical reason, that the output is put on the communication channel. A protocol described under this output style (and Yao's formalism), which we call a *shared-output* protocol, means that the protocol's output can be known to *both* Alice and Bob. We define $Q(f)$ as the CC for *one of them* to know the output since we want to regard one-way protocols as a special case of two-way protocols. Thus our $Q(f)$ may be smaller than the corresponding CC under shared-output protocols, but we can easily see that the gap is at most one qubit.

For the shared-output protocol, the following lemma, which was given by Yao [30] without proof and proved by Kremer [20], is quite strong.

Lemma 5 ([30] and [20]). *The final state of a shared-output quantum protocol for a Boolean function* f *on input* (x, y) *using* n *qubit of communication can be written as*

$$\sum_{i\in\{0,1\}^n} |A_i(x)\rangle|i_n\rangle|B_i(y)\rangle,$$

where $|A_i(x)\rangle$ *and* $|B_i(y)\rangle$ *are complex vectors of norm* ≤ 1*, and* i_n *is the* n*th bit of the index* i *and also the last bit of the communication channel (that is, the output bit).*

To see the intuitive meaning of this lemma might help understand the proof of Lemma 6 (our main lemma) more easily. There are two points: (i) The superposition consists of at most 2^n different states, independent of the size of the whole space. This allows us to consider only 2^{2n} ($2^n \times 2^n$) different combinations of vectors (and their inner product values) when calculating the trace of underlying density matrices whose size may be much larger. (ii) As one can see, a product of state $A_i(x)$ and state $B_j(y)$ exists only if $i = j$. This correspondence is translated into the same correspondence between the indices when calculating an inner product of a point and a hyperplane of the converted arrangement. A similar correspondence was also used in [7] for lower bounds of quantum exact and bounded-error protocols, and in [29] for tight lower bounds of quantum one-sided unbounded-error (which is referred as *nondeterministic*) protocols.

Let $k_f^* = \min(k_f, k_{f^t})$. Then here is our first main result.

Theorem 1. *For any Boolean function f, $\lceil \log \sqrt{k_f + 1/8} - 1/2 \rceil \leq Q(f) \leq \lceil \log \sqrt{k_f^* + 1} \rceil$.*

Theorem 1 induces the equality of two-way and one-way quantum CCs of a Boolean function within one qubit since we can verify that the difference of the upper bound from the lower bound is at most one for any integer $k_f > 0$. Recall that the difference between the two-way and one-way CCs is also at most one in the classical case [26]. For the proof of Theorem 1, it is enough to give the lower bound $Q(f) \geq \lceil \log \sqrt{k_f + 1/8 - 1/2} \rceil$ since $Q(f) \leq \min(Q^1(f), Q^1(f^t)) = \lceil \log \sqrt{k_f^* + 1} \rceil$. To do so, we relate quantum communication protocols to arrangements.

Lemma 6 (From quantum CC to arrangements). *An n-qubit shared-output protocol that computes a Boolean function f with success probability $1/2 + \epsilon$ can be converted to a $(2^{2n-1} - 2^{n-1})$-dimensional arrangement of magnitude at most 1 that realizes f with margin ϵ.*

Proof. Suppose that P is an n-qubit protocol for f. According to Lemma 5, we can write the final quantum state of P on input (x, y), $\boldsymbol{\rho}_{xy}$, as follows.

$$\boldsymbol{\rho}_{xy} = \sum_{i,j \in \{0,1\}^n} |A_i(x)\rangle|i_n\rangle|B_i(y)\rangle\langle A_j(x)|\langle j_n|\langle B_j(y)| = \boldsymbol{\rho}^0_{xy} + \boldsymbol{\rho}^1_{xy} + \widetilde{\boldsymbol{\rho}}_{xy},$$

where

$$\boldsymbol{\rho}^0_{xy} = \sum_{i,j \in \{0,1\}^n \text{ and } i_n = j_n = 0} |A_i(x)\rangle|0\rangle|B_i(y)\rangle\langle A_j(x)|\langle 0|\langle B_j(y)|,$$

$$\boldsymbol{\rho}^1_{xy} = \sum_{i,j \in \{0,1\}^n \text{ and } i_n = j_n = 1} |A_i(x)\rangle|1\rangle|B_i(y)\rangle\langle A_j(x)|\langle 1|\langle B_j(y)|,$$

and $\widetilde{\boldsymbol{\rho}}_{xy} = \boldsymbol{\rho}_{xy} - \boldsymbol{\rho}^0_{xy} - \boldsymbol{\rho}^1_{xy}$ such that $\mathrm{Tr}(\boldsymbol{\rho}_{xy}) = \mathrm{Tr}(\boldsymbol{\rho}^0_{xy}) + \mathrm{Tr}(\boldsymbol{\rho}^1_{xy}) = 1$. Note that $\mathrm{Tr}(\boldsymbol{\rho}^0_{xy})$ (resp. $\mathrm{Tr}(\boldsymbol{\rho}^1_{xy})$) is the probability that the output of P is 0 (resp. 1). By basic properties of the trace [25], $\mathrm{Tr}(\boldsymbol{\rho}^0_{xy})$ can be written as follows: $|m_A\rangle$ and $|m_B\rangle$ are the computational base of Alice's and Bob's spaces, respectively, and $b \in \{0, 1\}$. Then,

$$\begin{aligned}
\mathrm{Tr}(\boldsymbol{\rho}^0_{xy}) &= \sum_{m_A, b, m_B} \langle m_A|\langle b|\langle m_B|\boldsymbol{\rho}^0_{xy}|m_A\rangle|b\rangle|m_B\rangle \\
&= \sum_{m_A, m_B} \sum_{i,j \in \{0,1\}^{n-1}} \langle m_A|\langle m_B|(|A_{i0}(x)\rangle|B_{i0}(y)\rangle\langle A_{j0}(x)|\langle B_{j0}(y)|)|m_A\rangle|m_B\rangle \\
&= \sum_{i,j \in \{0,1\}^{n-1}} \sum_{m_A, m_B} \langle A_{j0}(x)|\langle B_{j0}(y)||m_A\rangle|m_B\rangle\langle m_A|\langle m_B||A_{i0}(x)\rangle|B_{i0}(y)\rangle \\
&= \sum_{i,j \in \{0,1\}^{n-1}} \langle A_{j0}(x)|A_{i0}(x)\rangle\langle B_{j0}(y)|B_{i0}(y)\rangle,
\end{aligned}$$

where the last equation holds since $\sum_{m_A,m_B} |m_A\rangle|m_B\rangle\langle m_A|\langle m_B| = I$ (completeness relation). Now, let us define the following vectors $\mathbf{a}(x) \in \mathbb{C}^{2^{2n-2}}$ and $\mathbf{b}(y) \in \mathbb{C}^{2^{2n-2}+1}$.

$$(\mathbf{a}(x))_k = (\mathbf{a}(x))_{ij} = \langle A_{j0}(x)|A_{i0}(x)\rangle,$$
$$(\mathbf{b}(y))_k = (\mathbf{b}(y))_{ij} = \langle B_{j0}(y)|B_{i0}(y)\rangle \text{ for } i,j \in \{0,1\}^{n-1}, \quad (\mathbf{b}(y))_{2^{2n-2}+1} = 1/2,$$

where the index $k \in [2^{2n-2}]$ naturally corresponds to the index $ij \in \{0,1\}^{2n-2}$. Since P computes $f(x,y)$ with success probability $1/2+\epsilon$, $\mathrm{Tr}(\boldsymbol{\rho}^0_{xy}) \geq 1/2+\epsilon$ if $f(x,y)=0$ and $\leq 1/2-\epsilon$ if $f(x,y)=1$. Thus, the points $\mathbf{a}(x)$ and hyperplanes $\mathbf{b}(y)$ can be considered as an arrangement that "realizes" f but they are in complex space. Fortunately, one can find an arrangement in $\mathbb{R}^{2^{2n-1}}$ that realizes f from the above arrangement by noticing that $\mathrm{Tr}(\boldsymbol{\rho}^0_{xy})$ is always real. Namely,

$$\begin{aligned}
\mathrm{Tr}(\boldsymbol{\rho}^0_{xy}) &= \sum_{i,j\in\{0,1\}^{n-1}} \langle A_{j0}(x)|A_{i0}(x)\rangle\langle B_{j0}(y)|B_{i0}(y)\rangle \;=\; \sum_{k\in[2^{2n-2}]} (\mathbf{a}(x))_k(\mathbf{b}(y))_k \\
&= \sum_{k\in[2^{2n-2}]} \mathrm{Re}\left((\mathbf{a}(x))_k(\mathbf{b}(y))_k\right) \\
&= \sum_{k\in[2^{2n-2}]} (\mathrm{Re}(\mathbf{a}(x))_k\mathrm{Re}(\mathbf{b}(y))_k - \mathrm{Im}(\mathbf{a}(x))_k\mathrm{Im}(\mathbf{b}(y))_k) \\
&= \sum_{k\in[2^{2n-1}]} (\mathbf{a}'(x))_k(\mathbf{b}'(y))_k, \qquad (1)
\end{aligned}$$

where

$$(\mathbf{a}'(x))_{2k-1} = \mathrm{Re}(\mathbf{a}(x))_k, \quad (\mathbf{a}'(x))_{2k} = -\mathrm{Im}(\mathbf{a}(x))_k,$$
$$(\mathbf{b}'(y))_{2k-1} = \mathrm{Re}(\mathbf{b}(x))_k, \quad (\mathbf{b}'(y))_{2k} = \mathrm{Im}(\mathbf{b}(x))_k, \text{ for } k \in [2^{2n-2}],$$

and we set $(\mathbf{b}'(y))_{2^{2n-1}+1} = 1/2$. Now by Eq.(1), the arrangement of points $\mathbf{a}'(x)$ and hyperplanes $\mathbf{b}'(y)$ realizes f with margin ϵ. Also, it is easy to see that its magnitude is at most 1. Furthermore, since $\langle A_{i0}(x)|A_{i0}(x)\rangle$ and $\langle B_{j0}(y)|B_{j0}(y)\rangle$ are already real, the dimension of the above arrangement can be reduced from 2^{2n-1} to $2^{2n-1}-2^{n-1}$.

Proof of Theorem 1. Let $n = Q(f)$. As mentioned before Lemma 5, there exists an $(n+1)$-qubit shared-output protocol that computes f with success probability larger than $1/2$. By Lemma 6, we can obtain a $(2^{2n+1}-2^n)$-dimensional arrangement realizing f. Thus $k_f \leq 2(2^n)^2 - 2^n$. By solving the quadratic inequality on 2^n, $Q(f) = n \geq \lceil \log(\sqrt{8k_f+1}+1)\rceil - 2$. The righthand side equals to $\lceil \log\sqrt{8k_f+1}\rceil - 2 = \lceil \log\sqrt{k_f+1/8} - 1/2\rceil$ by a simple consideration on rounding reals, and hence we obtain the desired lower bound of $Q(f)$. On the contrary, it was proven that $Q^1(f) = \lceil \log\sqrt{k_f+1}\rceil$ [14]. Since $Q(f) \leq \min(Q^1(f), Q^1(f^t))$ (by our definition mentioned before Lemma 5), we obtain the desired upper bound. These complete the proof.

4 Simultaneous Message Passing Models

The simultaneous message passing (SMP) model is the following three-party communication model: Alice and Bob have their inputs x and y, respectively, but they have no interaction at all. The third party with no access to input, called the *referee*, must compute a Boolean function $f(x, y)$ with the help of two messages sent from Alice and Bob. For such a model, the corresponding CC are defined similarly to two-way or one-way CCs.

We give quite tight characterizations of unbounded-error SMP CCs, $Q^{||}(f)$ and $C^{||}(f)$. First, we show the characterization of $Q^{||}(f)$ via k_f, which also implies that $Q^{||}(f)$ is the same as the sum of $Q^1(f)$ and $Q^1(f^t)$ up to two qubits.

Theorem 2. *For any Boolean function f, $Q^1(f) + Q^1(f^t) \leq Q^{||}(f) \leq Q^1(f) + Q^1(f^t) + 2$. In particular,*

$$\lceil \log \sqrt{k_f + 1} \rceil + \lceil \log \sqrt{k_{f^t} + 1} \rceil \leq Q^{||}(f) \leq 2 \lceil \log \sqrt{k_f^* + 2} \rceil.$$

Proof. For lower bound, $Q^1(f) + Q^1(f^t) \leq Q^{||}(f)$ is obtained by considering the relation between one-way communication models and SMP models: In the SMP model, Alice must send at least $Q^1(f)$ qubits to the referee. Otherwise, the number of qubits that she sends to the referee would be $m < Q^1(f)$, and then we can construct an m-qubit one-way protocol from Alice to Bob by regarding the referee and Bob as the same party, which contradicts the definition of $Q^1(f)$. Similarly Bob must send at least $Q^1(f^t)$ qubits. Since $Q^1(f) = \lceil \log \sqrt{k_f + 1} \rceil$ for any f, we obtain $\lceil \log \sqrt{k_f + 1} \rceil + \lceil \log \sqrt{k_{f^t} + 1} \rceil \leq Q^{||}(f)$, and $2 \lceil \log \sqrt{k_f^* + 2} \rceil \leq Q^1(f) + Q^1(f^t) + 2$.

What remains to do is to show the upper bound $Q^{||}(f) \leq 2 \lceil \log \sqrt{k_f^* + 2} \rceil$. For this purpose, we can use *quantum fingerprinting* introduced in [5]. That is, Alice's input x and Bob's y are encoded into two quantum states $\boldsymbol{\rho}_x$ and $\boldsymbol{\rho}_y$, respectively, and the referee uses the controlled SWAP (C-SWAP) test. The difference from the standard quantum fingerprinting such as [5,31,13] is that we use mixed states for encoding. (The C-SWAP test for mixed states are also used in [18] for quantum Merlin-Arthur games.)

We assume $k_f \leq k_{f^t}$ and show $Q^{||}(f) \leq 2 \lceil \log \sqrt{k_f + 2} \rceil$. (The case of $k_f > k_{f^t}$ is similarly shown.) Let $d = k_f$. Then there is an arrangement of points $\boldsymbol{p}_x = (p_i^x) \in \mathbb{R}^d$ and hyperplanes $\boldsymbol{h}_y = (h_i^y) \in \mathbb{R}^{d+1}$ that realizes f. Let $n = \lceil \log \sqrt{d + 2} \rceil$ and $N = 2^n$. Also, for each x, define $\boldsymbol{q}_x = (q_i^x) \in \mathbb{R}^{d+1}$ as $q_1^x = p_1^x, \ldots, q_d^x = p_d^x$, $q_{d+1}^x = -1$. By Lemma 3, for each $\boldsymbol{q}_x$ and $\boldsymbol{h}_y$ we can obtain n-qubit states $\boldsymbol{\rho}(\boldsymbol{q}_x) = \frac{1}{N} \left(\boldsymbol{I} + \sqrt{\frac{N(N-1)}{2}} \sum_{i=1}^{d+1} \left(\frac{q_i^x}{|\boldsymbol{q}_x|(N-1)} \right) \boldsymbol{\lambda}_i \right)$ and $\boldsymbol{\rho}(\boldsymbol{h}_y) = \frac{1}{N} \left(\boldsymbol{I} + \sqrt{\frac{N(N-1)}{2}} \sum_{i=1}^{d+1} \left(\frac{h_i^y}{|\boldsymbol{h}_y|(N-1)} \right) \boldsymbol{\lambda}_i \right)$. Then, we consider the following SMP quantum protocol: (1) Alice and Bob send the referee $\boldsymbol{\rho}(\boldsymbol{q}_x)$ and $\boldsymbol{\rho}(\boldsymbol{h}_y)$, respectively. (2) The referee outputs the bit obtained by the C-SWAP test on the

pair of the quantum states $(\boldsymbol{\rho}(\boldsymbol{q}_x), \boldsymbol{\rho}(\boldsymbol{h}_y))$ with probability $\alpha = \frac{1}{2}\left(\frac{1}{2} + \frac{1}{2N}\right)^{-1}$, and otherwise outputs 1 with probability $1 - \alpha$. Note that the C-SWAP test produces output 0 with probability $\frac{1}{2} + \frac{1}{2}\mathrm{Tr}(\boldsymbol{\rho}(\boldsymbol{q}_x)\boldsymbol{\rho}(\boldsymbol{h}_y))$ [5,18]. Thus, the referee outputs 0 with probability

$$\begin{aligned}
\alpha\left(\frac{1}{2} + \frac{1}{2}\mathrm{Tr}(\boldsymbol{\rho}(\boldsymbol{q}_x)\boldsymbol{\rho}(\boldsymbol{h}_y))\right) &= \frac{1}{2}\left(\frac{1}{2} + \frac{1}{2N}\right)^{-1}\left(\frac{1}{2} + \frac{1}{2N} + \frac{N-1}{2N}\sum_{i=1}^{d+1}\frac{q_i^x h_i^y}{|\boldsymbol{q}_x||\boldsymbol{h}_y|(N-1)^2}\right) \\
&= \frac{1}{2} + \frac{1}{4N|\boldsymbol{q}_x||\boldsymbol{h}_y|(N-1)}\left(\frac{1}{2} + \frac{1}{2N}\right)^{-1}\left(\sum_{i=1}^{d} p_i^x h_i^y - h_{d+1}^y\right) \\
&= \begin{cases} > 1/2 \text{ if } f(x,y) = 0 \\ < 1/2 \text{ if } f(x,y) = 1. \end{cases}
\end{aligned}$$

Hence $Q^{||}(f) \leq 2n = 2\lceil \log \sqrt{d+2} \rceil$.

Moreover, we can also show a similar result in the classical setting.

Theorem 3. *For any Boolean function f, $C^1(f) + C^1(f^t) \leq C^{||}(f) \leq C^1(f) + C^1(f^t) + 1$. In particular,*

$$\lceil \log(k_f + 1) \rceil + \lceil \log(k_{f^t} + 1) \rceil \leq C^{||}(f) \leq \lceil \log(k_f^* + 1) \rceil + \lceil \log(k_f^* + 2) \rceil.$$

5 Weakly Unbounded-Error Communication Complexity

Finally we give several relations among the weakly unbounded-error CCs. For this purpose, we need Lemmas 1, 6 and 7 to consider the bias of the success probability explicitly when converting protocols to arrangement, and vice versa.

Theorem 4. *The following relations hold for any Boolean function f: (1) $C_w(f) \leq C_w^1(f) \leq 3Q_w(f) + O(1)$. (2) $Q_w^1(f) \leq 2Q_w(f) + O(1)$.*

Proof. 1) By the definition of $Q_w(f)$, there is a quantum protocol P such that $Q_w(f) = C_P + \lceil \log 1/\epsilon_P \rceil$ where C_P and $1/2 + \epsilon_P$ are the communication cost and the success probability of P, respectively. By Lemma 6, we can obtain a $(2^{2C_P-1} - 2^{C_P-1})$-dimensional arrangement of magnitude at most 1 with margin ϵ_P from P. By Lemma 1, we have a $2C_P$-bit one-way protocol that computes f with probability $\geq 1/2 + \epsilon_P/(2\sqrt{2^{2C_P-1}})$. This implies that $C_w^1(f) \leq 2C_P + \lceil \log(2\sqrt{2^{2C_P-1}}/\epsilon_P) \rceil$, which is at most $3C_P + \lceil \log 1/\epsilon_P \rceil + O(1) \leq 3Q_w(f) + O(1)$.

2) The proof idea is similar to 1). The difference from 1) is to construct a desired protocol from the arrangement. To this end, we use the following lemma, whose proof is omitted, that convert arrangements to one-way quantum CC. The proof follows from carefully transforming points and hyperplanes, with appropriate shrinking and shifting factors, to quantum states (by Lemma 3) and measurements (by Lemma 4), respectively. The success probabilities of resulting protocols then follows from Lemma 5 of [14].

Lemma 7 (From arrangements to quantum CC). *Each d-dimensional arrangement of magnitude at most 1 realizing f with margin μ can be converted into an $n = \lceil \log \sqrt{d+1} \rceil$ qubit one-way protocol that computes f with success probability at least $1/2 + \alpha\mu$ where $\alpha = \frac{\sqrt{2}-1}{2^{n+1/2}}$.*

Now we give the proof of Theorem 4 (2). Take a quantum protocol P such that $Q_w(f) = C_P + \lceil \log 1/\epsilon_P \rceil$ where C_P and $1/2 + \epsilon_P$ are the communication cost and the success probability of P, respectively. By Lemma 6, we can obtain a $(2^{2C_P-1} - 2^{C_P-1})$-dimensional arrangement of magnitude at most 1 with margin ϵ_P from P. By Lemma 7, we have a one-way quantum protocol for f using at most C_P qubits such that its success probability is $1/2+\Omega(\epsilon_P/2^{C_P})$. This implies that $Q^1_w(f) \leq 2C_P + \lceil \log 1/\epsilon_P \rceil + O(1) \leq 2Q_w(f) + O(1)$.

Similar to the proof of Theorem 4, using the proofs of Theorems 2 and 3 we can also show: $Q^{||}_w(f) \leq 4Q_w(f) + O(1)$ and $C^{||}_w(f) \leq 9Q_w(f) + O(1)$.

Acknowledgements

We would like to acknowledge Francois Le Gall for insightful discussion on the two-way model, and Prof. Hiroshi Imai of University of Tokyo and ERATO-SORST project for partial support that enabled us to have a useful face-to-face discussion with quantum computation and information researchers in Japan while writing the paper.

References

1. Alon, N., Frankl, P., Rödl, V.: Geometrical realization of set systems and probabilistic communication complexity. In: Proc. 26th FOCS, pp. 277–280 (1985)
2. Ambainis, A., Nayak, A., Ta-shma, A., Vazirani, U.: Dense quantum coding and quantum finite automata. J. ACM 49, 496–511 (2002)
3. Babai, L., Frankl, P., Simon, J.: Complexity classes in communication complexity. In: Proc. 27th FOCS, pp. 303–312 (1986)
4. Babai, L., Kimmel, P.: Randomized simultaneous messages: solution of a problem of Yao in communication complexity. In: Proc. 12th CCC, pp. 239–246 (1997)
5. Buhrman, H., Cleve, R., Watrous, J., de Wolf, R.: Quantum fingerprinting. Phys. Rev. Lett. 87 Article no. 167902 (2001)
6. Buhrman, H., Vereshchagin, N., de Wolf, R.: On computation and communication with small bias. In: Proc. 22nd CCC, pp. 24–32 (2007)
7. Buhrman, H., de Wolf, R.: Communication Complexity Lower Bounds by Polynomials. In: Proc. 16th CCC, pp. 120–130, Also, cs.CC/9910010 (2001)
8. Forster, J.: A linear lower bound on the unbounded error probabilistic communication complexity. J. Comput. Syst. Sci. 65, 612–625 (2002)
9. Forster, J., Krause, M., Lokam, S.V., Mubarakzjanov, R., Schmitt, N., Simon, H.U.: Relations between communication complexity, linear arrangements, and computational complexity. In: Hariharan, R., Mukund, M., Vinay, V. (eds.) FST TCS 2001: Foundations of Software Technology and Theoretical Computer Science. LNCS, vol. 2245, pp. 171–182. Springer, Heidelberg (2001)
10. Forster, J., Simon, H.U.: On the smallest possible dimension and the largest possible margin of linear arrangements representing given concept classes. Theoret. Comput. Sci. 350, 40–48 (2006)
11. Gavinsky, D., Kempe, J., Kerenidis, I., Raz, R., de Wolf, R.: Exponential separations for one-way quantum communication complexity, with applications to cryptography. In: Proc. 39th STOC, pp. 516–525, Also, quant-ph/0611209 (2007)

12. Gavinsky, D., Kempe, J., Regev, O., de Wolf, R.: Bounded-error quantum state identification and exponential separations in communication complexity. In: Proc. 38th STOC, pp. 594–603 (2006)
13. Gavinsky, D., Kempe, J., de Wolf, R.: Strengths and weaknesses of quantum fingerprinting. In: Proc. 21st CCC, pp. 288–298 (2006)
14. Iwama, K., Nishimura, H., Raymond, R., Yamashita, S.: Unbounded-error one-way classical and quantum communication complexity. In: Arge, L., Cachin, C., Jurdziński, T., Tarlecki, A. (eds.) ICALP 2007. LNCS, vol. 4596, pp. 110–121. Springer, Heidelberg (2007) Also, quant-ph/0706.3265
15. Jakóbczyk, L., Siennicki, M.: Geometry of Bloch vectors in two-qubit system. Phys. Lett. A 286, 383–390 (2001)
16. Kimura, G., Kossakowski, A.: The Bloch-vector space for N-level systems – the spherical-coordinate point of view. Open Sys. Information Dyn. 12, 207–229 (2005)
17. Klauck, H.: Lower bounds for quantum communication complexity. SIAM J. Comput. 37, 20–46 (2007)
18. Kobayashi, H., Matsumoto, K., Yamakami, T.: Quantum Merlin-Arthur proof systems: Are multiple Merlins more helpful to Arthur? In: Ibaraki, T., Katoh, N., Ono, H. (eds.) ISAAC 2003. LNCS, vol. 2906, pp. 189–198. Springer, Heidelberg (2003)
19. Kossakowski, A.: A class of linear positive maps in matrix algebras. Open Sys. Information Dyn. 10, 213–220 (2003)
20. Kremer, I.: Quantum Communication. Master's Thesis, The Hebrew Univ. of Jerusalem (1995)
21. Kushilevitz, E., Nisan, N.: Communication Complexity. Cambridge (1997)
22. Linial, N., Shraibman, A.: Learning complexity vs. communication complexity (2006), Manuscript Available at `http://www.cs.huji.ac.il/~nati/`
23. Linial, N., Shraibman, A.: Lower bounds in communication complexity based on factorization norms. In: Proc. 39th STOC, pp. 699–708 (2007)
24. Newman, I., Szegedy, M.: Public vs. private coin flips in one-round communication games. In: Proc. 28th STOC, pp. 561–570 (1996)
25. Nielsen, M.A., Chuang, I.L.: Quantum Computation and Quantum Information. Cambridge (2000)
26. Paturi, R., Simon, J.: Probabilistic communication complexity. J. Comput. Syst. Sci. 33, 106–123 (1986) Preliminary version appeared in Proc. 25th FOCS, pp. 118–126 (1984)
27. Raz, R.: Exponential separation of quantum and classical communication complexity. In: Proc. 31st STOC, pp. 358–367 (1999)
28. Sherstov, A.: Halfspace matrices. In: Proc. 22nd CCC, pp. 83–95 (2007)
29. de Wolf, R.: Nondeterministic quantum query and communication complexities. SIAM J. Comput. 32, 681–699 (2003)
30. Yao, A.C.-C.: Quantum circuit complexity. In: Proc. 34th FOCS, pp. 352–360 (1993)
31. Yao, A.C.-C.: On the power of quantum fingerprinting. In: Proc. 35th STOC, pp. 77–81 (2003)

A Spectral Method for MAX2SAT in the Planted Solution Model

Masaki Yamamoto

Graduate School of Informatics, Kyoto University, Kyoto, 606-8501, Japan
masaki.yamamoto@kuis.kyoto-u.ac.jp

Abstract. We propose an algorithm using a spectral method, and analyze its average-case performance for MAX2SAT in the planted solution model. In [16], they proposed a distribution $\mathcal{G}_{n,p,r}$ for MAX2SAT in the planted solution model, as well as a message-passing algorithm. They showed that it solves, **whp**, MAX2SAT on $\mathcal{G}_{n,p,r}$ for rather dense formulas, i.e., the expected number of clauses is $\Omega(n^{1.5}\sqrt{\log n})$. In this paper, we propose an algorithm using a spectral method and a variant of message-passing algorithms, and show that it solves, **whp**, MAX2SAT on $\mathcal{G}_{n,p,r}$ for sparser formulas, i.e., the expected number of clauses is $\Omega(n \log n)$.

1 Introduction

In this paper, we propose an algorithm using a spectral method, and analyze its average-case performance for MAX2SAT in the planted solution model.

MAX2SAT asks, given a 2-CNF formula, what the truth assignment satisfying as many clauses as possible is. It is one of the basic NP-hard problems. Furthermore, there is no polynomial time approximation algorithm that finds an assignment with a ratio better than 0.955 unless P=NP [9]. Therefore, it is significant to define a suitable distribution over 2-CNF formulas, and to quest for efficient algorithms that solve MAX2SAT well *on average* for the distribution. In fact, there are several average-case analyses for the other NP-hard problems: graph coloring [1,5,2,7], graph partitioning [8,14,3,4], and 3-SAT [12,13,10], while it is not enough for MAX2SAT.

In [16], they proposed a distribution for MAX2SAT in the planted solution model, as well as a message-passing algorithm for it. The distribution, denoted by $\mathcal{G}_{n,p,r}$, seems to be natural: choose a planted assignment on n variables uniformly at random, say, the all-true assignment, and then independently generate clauses as follows: for every pair (x, y) of variables, include clauses which has literals with the same polarities, i.e., $(x \vee y)$ and $(\overline{x} \vee \overline{y})$, with probability r, and the other types of clauses with probability p. (See section 2 for a formal definition.) They showed that the planted assignment is optimal with high probability if $p, r = \Omega(\log n/n)$ with $p \geq 4r$. The message-passing algorithm they proposed, given a 2-CNF formula $I \in \mathcal{G}_{n,p,r}$, first constructs a directed graph from I in the standard manner. Next, it arbitrarily chooses a vertex u of the graph, and makes u believe that u is assigned true by the planted assignment. Suppose

T. Tokuyama (Ed.): ISAAC 2007, LNCS 4835, pp. 112–123, 2007.

that this "belief" is correct. Then, u propagates that belief to its neighbors v along edge (u, v), and further v also propagates its belief to its neighbors, and so on. This propagation rule comes from the following idea: clause $(u \to v)$ is not satisfied if and only if u and v are assigned true and false respectively. This propagation procedure is repeated in parallel for each round, and repeated a finite number of rounds. Finally, it decides a truth value of each variable x according to two beliefs at vertices x and $\overline{x}$. They analyzed this message-passing algorithm with *two rounds*, and showed that it solves MAX2SAT on $\mathcal{G}_{n,p,r}$ with $p, r = \Omega(\sqrt{\log n/n})$ with high probability, that is, rather dense formulas.

1.1 Our Ideas

This two-round message-passing algorithm makes use of correct beliefs at only one vertex. On the other hand, using the well-studied spectral method [1,12], we can obtain correct beliefs at a $(1-\delta)$-fraction of variables for any (small) constant $\delta > 0$. In particular, [12] proposed an algorithm for 3-SAT using the spectral method in the similar way to [1]. He analyzed its average-case performance for distribution I_{n,p_1,p_2,p_3}, where we choose a planted assignment on n variables uniformly at random, and then we include in $I \in I_{n,p_1,p_2,p_3}$ each clause with exactly i literals satisfied by the planted assignment. Parameterizing as $p_1 = d/n^2$, $p_2 = \eta_2 d/n^2$, and $p_3 = \eta_3 d/n^2$, he showed a theorem (corollary 2.1 of [12]) that for any constants $0 \le \eta_2, \eta_3 \le 1$ except $(\eta_2, \eta_3) = (0, 1)$, there exist constants d_{min} and C such that for any $d \ge d_{min}$, after applying the spectral method to $I \in I_{n,p_1,p_2,p_3}$, at least $(1 - C/d)n$ variables of I are set correctly with high probability.

Once we are given correct beliefs at a $(1 - \delta)$-fraction of variables for some suitably small constant $\delta > 0$, it is not difficult to see that a variant of message-passing algorithms works well for *somewhat* dense instances: Given $I \in \mathcal{G}_{n,p,r}$ and an assignment t which agrees with the planted assignment at $(1 - \delta)n$ variables for a suitably small constant $\delta > 0$. Suppose that I is represented as a directed graph constructed in the standard manner. Then, we proceed as follows: for each variable x, count the numbers $W(x)$ and $W(\overline{x})$ of directed edges emanating from vertices assigned true by t into x and $\overline{x}$ respectively. If $W(x) \ge W(\overline{x})$, we redefine as $t(x) = 1$, and as $t(x) = 0$ otherwise. This procedure is done in parallel for each variable, and repeated $\log n$ rounds. By a counting argument, it is not difficult to see that the number of variables incorrectly assigned decreases by half for each round, and therefore the final assignment after $\log n$ rounds is consistent with the planted assignment with high probability. The following is our main result for this algorithm `spect-max2sat`:

Theorem 1. Let $p = c_p \log n/n$ and $r = c_r \log n/n$. Given $I \in \mathcal{G}_{n,p,r}$, for any sufficiently large constants c_p and c_r such that $c_p - c_r$ is also sufficiently large, `spect-max2sat`(I) outputs its planted assignment with high probability.

Note that the expected number of clauses generated from $\mathcal{G}_{n,p,r}$ with parameters stated in this theorem is $\Omega(n \log n)$ while it is $\Omega(n^{1.5}\sqrt{\log n})$ in [16].

1.2 Related Works

We mention about two papers [6] and [15] which studied the average-case performance for MAX2SAT. These average-case analyses are not in the planted solution model, but w.r.t. uniform distribution over instances. In [15], they proposed an expected polynomial time algorithm for *very sparse* instances, that is, the expected number of clauses is at most $n/2$. In [6], they proposed an expected polynomial time *approximation* algorithm for sparse instances, that is, the clauses/variables ratio is at least some constant.

2 Preliminaries

We assume that readers are familiar with CNF-formulas, in particular, 2-CNF formulas. Let I be a 2-CNF formula over X. We let $X = \{1, \ldots, n\}$ throughout this paper. Furthermore, we denote a set of negative literals of X by $\overline{X} = \{-1, \ldots, -n\}$. Let t be a truth assignment (an assignment for short) to X. We denote by 1 (resp. 0) truth value "true" (resp. "false"), and denote by $t(x) = 1$ (resp. $t(x) = 0$) that x is assigned 1 (resp. 0) by t. Furthermore, we denote by $\overline{t}$ its complement, that is, $\overline{t}(x) = 1 - t(x)$ for every $x \in X$.

Next, we define our distribution over instances of MAX2SAT by giving a procedure generating formulas. For generating a random 2-CNF formula I, we first choose an assignment (called a *planted assignment*, and denoted by ϕ throughout this paper) uniformly at random, then independently generate clauses according to the following probabilities: for $1 \leq i \leq j \leq n$,

$$\begin{aligned} \Pr\{(f(i) \vee -f(j)) \in I\} &= \Pr\{(-f(i) \vee f(j)) \in I\} \quad = p, \\ \Pr\{(f(i) \vee f(j)) \in I\} \quad &= \Pr\{(-f(i) \vee -f(j)) \in I\} = r, \end{aligned}$$

where $f(i) = i$ if variable i is assigned 1 by ϕ, and $f(i) = -i$ otherwise [1]. We denote this distribution by $\mathcal{G}_{n,p,r}$. We say that an event occurs *with high probability*, denoted by ***whp***, if the probability tends to one as $n \to \infty$.

Theorem 2. [Watanabe and Yamamoto [16]] Let $p = c_p \log n/n$ and $r = c_r \log n/n$ for any sufficiently large constants c_p, c_r. If $c_p \geq 4c_r$, then the optimal assignment of $I \in \mathcal{G}_{n,p,r}$ is identical to the planted assignment ϕ or $\overline{\phi}$ **whp**.

In this paper, we focus on $\mathcal{G}_{n,p,r}$ where $p \stackrel{\text{def}}{=} c_p \log n/n$ and $r \stackrel{\text{def}}{=} c_r \log n/n$ for any sufficiently large constants c_p, c_r such that $c_p - c_r$ is also sufficiently large.

3 Analysis of the Average-Case Performance

In this section, we present an algorithm `spect-max2sat` for MAX2SAT, which consists of two stages: (1) a spectral method, and (2) a $\log n$-round message-passing procedure, and then show that it solves MAX2SAT on $\mathcal{G}_{n,p,r}$ **whp**.

[1] For $i = j$, we define $(i \vee j)$ (resp. $(\overline{i} \vee \overline{j})$) as unit clause (i) (resp. $(\overline{i})$). Furthermore, we generate only $(i \vee \overline{i})$, and do not generate the other type $(\overline{i} \vee i)$.

Before giving a description of the algorithm, we define some notions and notations for it. Given a 2-CNF formula I over X, we define an *incidence matrix* A, where rows and columns are indexed by literals $1, \ldots, n, -1, \ldots, -n$ in this order, and the (a, b)-th entry of A is one if two literals corresponding to the ath row and the bth column simultaneously occur in some clause of I, and zero otherwise. Note that, since A is a $2n$-by-$2n$ symmetric matrix, it has $2n$ eigenvalues, denoted by $\lambda_1 \geq \cdots \geq \lambda_{2n}$, and an orthonormal basis of $2n$ eigenvectors, denoted by $\boldsymbol{v}^{(1)}, \ldots, \boldsymbol{v}^{(2n)}$.

Given a 2-CNF formula I over X, let $G = (V, E)$ be a directed graph constructed from I in the following manner: let $V_+ \stackrel{\text{def}}{=} X$, $V_- \stackrel{\text{def}}{=} \overline{X}$, and

$$V \stackrel{\text{def}}{=} V_+ \cup V_-, \quad E \stackrel{\text{def}}{=} \{(-i, j), (-j, i) : (i \vee j) \in I\}.$$

Let t be an assignment to V. A directed edge (j, i) for $i, j \in V$ is called a *warning edge* at i under t iff j is assigned 1 by t.

Now, we present our algorithm `spect-max2sat` for MAX2SAT in Figure 1. Here, observing that the expected number of occurrences of ℓ is $pn + rn$ for any

`spect-max2sat`(I)

step 0 Form the incidence matrix A from I. If there is a row having at least $2(pn + rn)$ 1s, then output `failure`.
step 1 Calculate the last eigenvector $\boldsymbol{v}^{(2n)}$ of A.
step 2 Obtain an assignment t according to $\boldsymbol{v} \stackrel{\text{def}}{=} \boldsymbol{v}^{(2n)}$ as follows: for $1 \leq i \leq n$, $t(i) = 1$ if $\boldsymbol{v}_i \geq 0$, and $t(i) = 0$ otherwise.
step 3 Construct the directed graph $G = (V, E)$ from I, and output the better assignment t and $\overline{t}$ by repeating the following procedure $\log n$ rounds:

for each variable $i \in X$ **do**
Count the numbers $W(i)$ and $W(-i)$ of warning edges at i and $-i$ respectively under t.
If $W(i) \geq W(-i)$, then update t to $t(i) = 1$, otherwise to $t(i) = 0$.
end-for-each

Fig. 1. Algorithm

literal ℓ of $I \in \mathcal{G}_{n,p,r}$, we note the following proposition which is easily obtained by Chernoff bounds [2].

Proposition 1. For any sufficiently large c_p, c_r, the probability that algorithm `spect-max2sat`(I) outputs `failure` for $I \in \mathcal{G}_{n,p,r}$ is $o(1)$.

In the following two subsections, we analyze the average-case performance of `spect-max2sat`(I) for $I \in \mathcal{G}_{n,p,r}$, and show that it outputs ϕ or $\overline{\phi}$ **whp**, therefore the optimal assignment of I **whp**.

[2] We can do it because we consider a somewhat dense instances, that is, the expected number of occurrences of any literal is $\Omega(\log n)$ while it is only a constant in [12].

3.1 Spectral Arguments

In this subsection, we show the following theorem in the similar way to [12].

Theorem 3. For any suitably small $\delta > 0$, and for any sufficiently large c_p and c_r (depending on δ) such that $c_p - c_r$ is also sufficiently large (depending on δ), the assignment obtained from step 2 of `spect-max2sat` disagrees with ϕ on at most δn variables **whp**.

We first define some key values, which depend on c_p and c_r. (See [12] for the intuition for these values.) Consider the following 2-dimensional system:

$$\begin{pmatrix} c_r & c_p \\ c_p & c_r \end{pmatrix} \begin{pmatrix} \beta \\ \gamma \end{pmatrix} = \alpha \begin{pmatrix} \beta \\ \gamma \end{pmatrix}$$

By an elementary calculation, we obtain its eigenvalues $\alpha = c_r \pm c_p$. For those two values, let

$$\alpha_+ \stackrel{\text{def}}{=} c_p + c_r > 0 \text{ and } \alpha_- \stackrel{\text{def}}{=} -(c_p - c_r) < 0.$$

Furthermore, two unit eigenvectors corresponding to α_+ and α_- are

$$\begin{pmatrix} \beta_+ \\ \gamma_+ \end{pmatrix} \stackrel{\text{def}}{=} \begin{pmatrix} 1/\sqrt{2} \\ 1/\sqrt{2} \end{pmatrix} \text{ and } \begin{pmatrix} \beta_- \\ \gamma_- \end{pmatrix} \stackrel{\text{def}}{=} \begin{pmatrix} 1/\sqrt{2} \\ -1/\sqrt{2} \end{pmatrix},$$

respectively. Note that β_- and γ_- have opposite signs. Let $\boldsymbol{v}_T$ and $\boldsymbol{v}_F$ be $2n$-dimensional vectors indexed by literals $1, \ldots, n, -1, \ldots, -n$ in this order, and $\boldsymbol{v}_T(\ell) = 1$ if literal ℓ is assigned 1 by ϕ, and $\boldsymbol{v}_T(\ell) = 0$ otherwise, and let $\boldsymbol{v}_F(\ell) = 1 - \boldsymbol{v}_T(\ell)$.

The following lemma holds, which is proved in the similar way to [12], which is also similar to [1].

Lemma 1. Given $I \in \mathcal{G}_{n,p,r}$. Let A be the incidence matrix for I, and $\lambda_1 \geq \cdots \geq \lambda_{2n}$ be the eigenvalues of A. Then, for any suitably small $\epsilon > 0$, and for any sufficiently large c_p and c_r (depending on ϵ) such that $c_p - c_r$ is also sufficiently large (depending on ϵ), the following holds **whp**:

(1) $\lambda_1 \geq ((1-\epsilon)\alpha_+) \log n$,
(2) $\lambda_{2n} \leq (1-2\epsilon)\alpha_- \log n$, and
(3) $|\lambda_i| \leq 4\sqrt{c_p \log n}$ for $2 \leq i \leq 2n-1$.

Proof. Throughout this proof, we fix $\epsilon > 0$ to be a suitably small constant. We make use of Rayleigh quotient principle that

$$(*): \quad \lambda_i = \min_{L} \max_{\boldsymbol{v} \in L,\, \boldsymbol{v} \neq \boldsymbol{0}} \frac{\boldsymbol{v}^{\mathbf{T}} A \boldsymbol{v}}{\boldsymbol{v}^{\mathbf{T}} \boldsymbol{v}} = \max_{L'} \min_{\boldsymbol{v} \in L',\, \boldsymbol{v} \neq \boldsymbol{0}} \frac{\boldsymbol{v}^{\mathbf{T}} A \boldsymbol{v}}{\boldsymbol{v}^{\mathbf{T}} \boldsymbol{v}},$$

where L (resp. L') ranges over all $(2n-i+1)$-dimensional (resp. i-dimensional) subspaces of R^{2n}.

Consider the following partition of A: $A[i,j]$ for $i,j \in \{T,F\}$, where $A[T,T]$ is a submatrix of A for literals assigned 1 by ϕ, and the other ones are similarly defined. We estimate the number of 1s of $A[i,j]$, that is, $\boldsymbol{v}_i^{\mathbf{T}} A \boldsymbol{v}_j$, which is easily proved by Chernoff bounds.

Proposition 2. For any sufficiently large c_p, c_r, the following holds **whp**: for $i, j \in \{T, F\}$,

$$(1-\epsilon)c_r n \log n \le \boldsymbol{v}_i^{\mathbf{T}} A \boldsymbol{v}_j \le (1+\epsilon)c_r n \log n \quad \text{for } i = j, \text{ and}$$
$$(1-\epsilon)c_p n \log n \le \boldsymbol{v}_i^{\mathbf{T}} A \boldsymbol{v}_j \le (1+\epsilon)c_p n \log n \quad \text{for } i \neq j.$$

For proving part (1), we apply $\boldsymbol{v}_+ \stackrel{\text{def}}{=} \beta_+ \boldsymbol{v}_T + \gamma_+ \boldsymbol{v}_F$ to the minmax version of $(*)$, and obtain the following **whp**:

$$\begin{aligned}
\boldsymbol{v}_+^T A \boldsymbol{v}_+ &= \beta_+^2 \boldsymbol{v}_T^{\mathbf{T}} A \boldsymbol{v}_T + 2\beta_+\gamma_+ \boldsymbol{v}_T^{\mathbf{T}} A \boldsymbol{v}_F + \gamma_+^2 \boldsymbol{v}_F^{\mathbf{T}} A \boldsymbol{v}_F \\
&\ge ((1-\epsilon) n \log n) \left(\beta_+^2 c_r + 2\beta_+\gamma_+ c_p + \gamma_+^2 c_r\right) \quad (\because\ \beta_+, \gamma_+ \ge 0) \\
&= ((1-\epsilon) n \log n) \left(\begin{pmatrix} \beta_+ & \gamma_+ \end{pmatrix} \begin{pmatrix} c_r & c_p \\ c_p & c_r \end{pmatrix} \begin{pmatrix} \beta_+ \\ \gamma_+ \end{pmatrix} \right) \\
&= ((1-\epsilon)\alpha_+)\, n \log n.
\end{aligned}$$

Since $v_+^{\mathbf{T}} v_+ = n$, we conclude that $\lambda_1 \ge ((1-\epsilon)\alpha_+) \log n$ **whp**. (We also have that $\lambda_1 \le ((1+\epsilon)\alpha_+) \log n$ **whp**.)

For proving part (2), we apply $L = \{t\boldsymbol{v}_- : t \in R\}$, to the minmax version of $(*)$, where $\boldsymbol{v}_- \stackrel{\text{def}}{=} \beta_- \boldsymbol{v}_T + \gamma_- \boldsymbol{v}_F$, and obtain the following **whp**:

$$(t\boldsymbol{v}_-)^T A (t\boldsymbol{v}_-) \le (t^2 n)(1-2\epsilon)\alpha_- \log n.$$

(See Appendix for a derivation of the above.) Since $(tv_-)^{\mathbf{T}}(tv_-) = t^2 n$, we conclude that $\lambda_{2n} \le (1-2\epsilon)\alpha_- \log n$ **whp**.

For proving (3), we need the following lemma [1]: Fix $z > 0$, say, $z = 1/2$. Let S denote the set of all vectors of length n, every coordinate of which is an integral multiple of $z/\sqrt{n}$, where the sum of coordinates is zero and l_2-norm is at most one. Let B be a random n-by-n 0/1-matrix, where each entry of B, randomly and independently, is one with probability d/n. Then:

Lemma 2. [Alon and Kahale [1]] If d exceeds a sufficiently large absolute constant, then **whp**, $|x^{\mathbf{T}} B y| = O(\sqrt{d})$ for every $x, y \in S$, for which $x_i = 0$ if the i-th row of B has more than $5d$ non-zero entries, and for which $y_i = 0$ if the i-th column of B has more than $5d$ non-zero entries.

We observe that the same statement above holds for a random n-by-n *symmetric* 0/1-matrix B, where each (a, b)-th entry for $1 \le a \le b \le n$, randomly and independently, is one with probability d/n. Then, we have the following proposition. (See Appendix for a proof.)

Proposition 3. For any unit vector $\boldsymbol{v}$ with $\boldsymbol{v}^{\mathbf{T}} \boldsymbol{v}_T = 0$ and $\boldsymbol{v}^{\mathbf{T}} \boldsymbol{v}_F = 0$, we have that $|\boldsymbol{v}^{\mathbf{T}} A \boldsymbol{v}| \le 4\sqrt{c_p \log n}$.

We need the following proposition, which is proved in the same way as [12]. (See Appendix for a proof.)

Proposition 4. For any sufficiently large c_p, c_r such that $c_p - c_r$ is also sufficiently large, the following holds **whp**:

$$\|(A - (\alpha_+ \log n)I)\boldsymbol{v}_+\|^2 \le (3\sqrt{\epsilon}\alpha_+ \log n)^2\|\boldsymbol{v}_+\|^2, \text{ and}$$
$$\|(A - (\alpha_- \log n)I)\boldsymbol{v}_-\|^2 \le (3\sqrt{\epsilon}\alpha_- \log n)^2\|\boldsymbol{v}_-\|^2$$

Now, we are ready to prove part (3): for λ_2 we apply $L = \{\boldsymbol{v} : \boldsymbol{v}^{\mathbf{T}}\boldsymbol{v}_+ = 0\}$ to the minmax version of $(*)$. Note that we can express $\boldsymbol{v} \in L$ as $t\boldsymbol{v}_- + \boldsymbol{w}$ for some $t \in R$ such that $\boldsymbol{w}^{\mathbf{T}}\boldsymbol{v}_+ = 0$ and $\boldsymbol{w}^{\mathbf{T}}\boldsymbol{v}_- = 0$. Thus, we have the following **whp**:

$$\begin{aligned}
\boldsymbol{v}^{\mathbf{T}}A\boldsymbol{v} &= (t\boldsymbol{v}_-)^{\mathbf{T}}A(t\boldsymbol{v}_-) + 2t\boldsymbol{w}^{\mathbf{T}}A\boldsymbol{v}_- + \boldsymbol{w}^{\mathbf{T}}A\boldsymbol{w} \\
&= (t\boldsymbol{v}_-)^{\mathbf{T}}A(t\boldsymbol{v}_-) + 2t\boldsymbol{w}^{\mathbf{T}}(A - (\alpha_+ \log n)I)\boldsymbol{v}_- + \boldsymbol{w}^{\mathbf{T}}A\boldsymbol{w} \\
&\le ((1-2\epsilon)\alpha_- \log n)t^2\|\boldsymbol{v}_-\|^2 \\
&\qquad +2t\|\boldsymbol{w}\|\|(A - (\alpha_+ \log n)I)\boldsymbol{v}_-\| + 4\sqrt{c_p \log n}\|\boldsymbol{w}\|^2 \\
&\le ((1-2\epsilon)\alpha_- \log n)t^2\|\boldsymbol{v}_-\|^2 \\
&\qquad +2t\|\boldsymbol{w}\|\left(3\sqrt{\epsilon}\alpha_+ \log n\right)\|\boldsymbol{v}_-\| + 4\sqrt{c_p \log n}\|\boldsymbol{w}\|^2 \\
&\le ((1-2\epsilon)\alpha_- \log n)\|\boldsymbol{v}\|^2 + \left(6\sqrt{\epsilon}\alpha_+ \log n\right)\|\boldsymbol{v}\|^2 + 4\sqrt{c_p \log n}\|\boldsymbol{v}\|^2 \\
&\le 4\sqrt{c_p \log n}\|\boldsymbol{v}\|^2, \quad (\because\ (1-2\epsilon)\alpha_- + 6\sqrt{\epsilon}\alpha_+ < 0)
\end{aligned}$$

which comes from $\|\boldsymbol{w}\| \le \|\boldsymbol{v}\|$ and $t\|\boldsymbol{v}_-\| \le \|\boldsymbol{v}\|$. Thus, we conclude that $\lambda_2 \le 4\sqrt{c_p \log n}$ **whp**.

For λ_{2n-1}, applying $L' = \{\boldsymbol{v} : \boldsymbol{v}^{\mathbf{T}}\boldsymbol{v}_- = 0\}$ to the maxmin-version of $(*)$, we obtain in the same way that $\lambda_{2n-1} \ge -4\sqrt{c_p \log n}$ **whp**. □

Using this lemma, we show the following lemma, from which we can obtain Theorem 3 as its corollary.

Lemma 3. Let λ_{2n} and $\boldsymbol{v}^{(2n)}$ be as defined in the beginning of section 3. Let $\boldsymbol{v}_- = \beta_-\boldsymbol{v}_T + \gamma_-\boldsymbol{v}_F$. Then, for any suitably small $\epsilon > 0$, and for any sufficiently large c_p and c_r (depending on ϵ) such that $c_p - c_r$ is sufficiently large (depending on ϵ), the sign of $\boldsymbol{v}^{(2n)}$ or $-\boldsymbol{v}^{(2n)}$ disagrees with $\boldsymbol{v}_-$ on at most $72\epsilon n$ coordinates **whp**.

Proof. We express $\boldsymbol{v}_-$ as a linear combination of orthonormal eigenvectors of A: that is, $\boldsymbol{v}_- = \sum_{i=1}^{2n} c_i\boldsymbol{v}^{(i)}$. Then, we have that

$$\begin{aligned}
\|((\alpha_- \log n)I - A)\boldsymbol{v}_-\|^2 &= \left\|\sum_{i=1}^{2n}(\alpha_- \log n - \lambda_i)c_i\boldsymbol{v}^{(i)}\right\|^2 \\
= \sum_{i=1}^{2n}(\alpha_- \log n - \lambda_i)^2c_i^2 &\ge \sum_{i=1}^{2n-1}(\alpha_- \log n - \lambda_i)^2c_i^2 \ge ((\alpha_-/2)\log n)^2\sum_{i=1}^{2n-1}c_i^2,
\end{aligned}$$

where the last inequality comes from the previous lemma. On the other hand, from Proposition 4, we have the following **whp**:

$$\|((\alpha_- \log n)I - A)\boldsymbol{v}_-\|^2 \le 9\epsilon(\alpha_- \log n)^2 n$$

Thus, we have the following **whp**:

$$\sum_{i=1}^{2n-1} c_i^2 \leq \frac{9\epsilon(\alpha_- \log n)^2 n}{((\alpha_-/2)\log n)^2} = 36\epsilon n.$$

Recall that each coordinate of $\boldsymbol{v}_-$ has $1/\sqrt{2}$ in absolute value. From this fact and $\sum_{i=1}^{2n-1} c_i^2 \leq 36\epsilon n$, it is not difficult to derive this lemma: Let $\widetilde{\boldsymbol{v}} \stackrel{\text{def}}{=} \sum_{i=1}^{2n-1} c_i \boldsymbol{v}^{(i)}$, and let S be a set of coordinates of $\boldsymbol{v}^{(2n)}$ on which the sign of $\boldsymbol{v}^{(2n)}$ disagrees with $\boldsymbol{v}_-$. Since $c_{2n}\boldsymbol{v}^{(2n)} = \boldsymbol{v}_- - \widetilde{\boldsymbol{v}}$, each coordinate $i \in S$ of $\widetilde{\boldsymbol{v}}$ must exceed $1/\sqrt{2}$ in absolute value. Thus, we have $|S| \leq 72\epsilon n$ from the following:

$$36\epsilon n \geq \sum_{i=1}^{2n-1} c_i^2 = \left\| \sum_{i=1}^{2n-1} c_i \boldsymbol{v}^{(i)} \right\|^2 = \|\widetilde{\boldsymbol{v}}\|^2 \geq \frac{1}{2}|S|.$$

□

3.2 Counting Arguments

In this subsection, we show that the assignment after applying $\log n$ times the procedure in step 3 is identical to ϕ **whp**, which together with Theorem 3 proves Theorem 1.

Let $G = (V, E)$, V_+, and V_- be as defined in the beginning of this section. Let $S_+^{(k)} \subset V_+$ (resp. $S_-^{(k)} \subset V_-$) be a subset of positive literals (resp. negative literals) which are incorrectly assigned after the kth ($0 \leq k \leq \log n$) round. (Note that $i \in S_+^{(k)}$ iff $-i \in S_-^{(k)}$ for any i and k.) We will show the following lemma, from which we can easily conclude that the assignment after applying $\log n$ times the procedure in step 3 is identical to ϕ **whp**.

Lemma 4. Let $0 < \delta \ll 1$ be a suitably small constant. For any sufficiently large c_p and c_r (depending on δ) such that $c_p - c_r$ is also sufficiently large (depending on δ), and for every round k ($1 \leq k \leq \log n$), if $|S_+^{(k-1)}| \leq \delta n$, then $|S_+^{(k)}| \leq |S_+^{(k-1)}|/2$ **whp**.

We will show this lemma by assuming w.l.o.g. that $\phi = 1^n$. We here call any directed edge (u, v) an *implied edge at* v. We first note the following proposition which is obtained by Chernoff bounds.

Proposition 5. For sufficiently large c_p, c_r, we have the following **whp**: for every vertex $v \in V_+$ (resp. $v' \in V_-$), (1) the number of implied edges within V_+ at v (resp. within V_- at v') is within $(1 \pm 0.1)c_p \log n$, and (2) the number of implied edges (u, v) at v (resp. (u', v') at v') with $u \in V_-$ (resp. $u' \in V_+$) is within $(1 \pm 0.1)c_r \log n$.

We here let $d \stackrel{\text{def}}{=} c_p \log n$. Consider a kth ($1 \leq k \leq \log n$) round of step 3. We claim that for every vertex $i \in V_+$, if $i \in S_+^{(k)}$, then at least one of the following two surely occurs: (1) the number of implied edges (j, i) at i with $j \in S_+^{(k-1)}$ is at least $d/3$, and (2) the number of implied edges $(j, -i)$ at $-i$ with

$j \in S_-^{(k-1)}$ is at least $d/3$. This is due to the following observation: consider the contraposition of the claim, i.e., the numbers of such implied edges at i and $-i$ respectively are both less than $d/3$. Then, from the above proposition, we have that $W(i) \geq 0.9d - d/3$ and $W(-i) \leq d/3 + 1.1c_r \log n$ **whp**, therefore

$$\begin{aligned} W(i) - W(-i) &\geq 0.9d - 2d/3 - 1.1c_r \log n \geq 0.1d - 1.1c_r \log n \\ &= (0.1c_p - 1.1c_r) \log n, \end{aligned}$$

i.e., $W(i) \geq W(-i)$ **whp** if $c_p - c_r$ is sufficiently large. This means $i \notin S_+^{(k)}$. Using this claim as well as the following lemma, we prove Lemma 4, regarding U, W as $S_+^{(k-1)}$, $S_+^{(k)}$ respectively.

Lemma 5. We have the following **whp**: there are no two subsets U and W of V_+ such that $|U| \leq \delta n$, $|W| = |U|/2$, and every vertex w of W has at least $d/3$ implied edges (u, w) with $u \in U$.

Proof. Fix arbitrarily U and W such that $|U| \leq \delta n$ and $|W| = |U|/2$. Observe that if every vertex w of W has at least $d/3$ implied edges (u, w) with $u \in U$, then $|(U \times W) \cap E| \geq d|W|/3$. Note that all edges of $U \times W$ are mutually independent. Thus, the probability that such two subsets exist is at most

$$\sum_{i=1}^{\delta n/2} \binom{n}{i} \binom{n}{2i} \binom{2i^2}{di/3} \left(\frac{d}{n}\right)^{di/3} = O\left(\frac{1}{n^{\Omega(d)}}\right),$$

where the left-hand-side of the above is obtained from the identical calculation to [1]. □

4 Conclusion

We have given an algorithm using the spectral method and the $\log n$-round message-passing procedure, and shown that it solves, **whp**, MAX2SAT on $\mathcal{G}_{n,p,r}$ for somewhat dense formulas, i.e., the expected number of clauses is $\Omega(n \log n)$. An obvious future work is to analyze it for formulas with linear density. In [16], they also proposed another distribution that such sparse formulas are generated. The procedure defining this distribution is rather different from that of $\mathcal{G}_{n,p,r}$: it generates clauses with a certain dependency. Therefore, it is not obvious that the same algorithm can be applied to such a distribution, in particular, the part of the spectral method.

References

1. Alon, N., Kahale, N.: A spectral technique for coloring random 3-colorable graphs. SIAM J. Comput. 26(6), 1733–1748 (1997)
2. Böttcher, J.: Coloring sparse random k-colorable graphs in polynomial expected time. In: Jedrzejowicz, J., Szepietowski, A. (eds.) MFCS 2005. LNCS, vol. 3618, pp. 156–167. Springer, Heidelberg (2005)

3. Coja-Oghlan, A.: A spectral heuristic for bisecting random graphs. In: Proceedings of 16th ACM-SIAM Symp. on Discrete Algorithms (SODA 2005), pp. 850–859 (2005)
4. Coja-Oghlan, A.: An Adaptive Spectral Heuristic for Partitioning Random Graphs. In: Bugliesi, M., Preneel, B., Sassone, V., Wegener, I. (eds.) ICALP 2006. LNCS, vol. 4052, pp. 691–702. Springer, Heidelberg (2006)
5. Coja-Oghlan, A., Taraz, A.: Colouring Random Graphs in Expected Polynomial Time. In: Alt, H., Habib, M. (eds.) STACS 2003. LNCS, vol. 2607, pp. 511–522. Springer, Heidelberg (2003)
6. Coja-Oghlan, A., Goerdt, A., Lanka, A., Schädlich, F.: Techniques from combinatorial approximation algorithms yield efficient algorithms for random $2k$-SAT. Theoretical Computer Science 329, 1–45 (2004)
7. Coja-Oghlan, A., Krivelevich, M., Vilenchik, D.: Why almost all k-colorable graphs are easy. In: Proceedings of 24th International Symp. on Theoretical Aspects of Computer Science (STACS 2007), pp. 121–132 (2007)
8. Condon, A., Karp, R.M.: Algorithms for graph partitioning on the planted partition model. Random Struct. Algorithms 18(2), 116–140 (2001)
9. Håstad, J.: Some optimal inapproximability results. J. of the ACM 48, 798–859 (2001)
10. Feige, U., Mossel, E., Vilenchik, D.: Complete convergence of message passing algorithms for some satisfiability problems. In: Proceedings of APPROX-RANDOM 2006, pp. 339–350 (2006)
11. Feige, U., Vilenchik, D.: A local search algorithm for 3SAT. Technical report of the Weizmann Institute of science (2004)
12. Flaxman, A.: A spectral technique for random satisfiable 3CNF formulas. In: Proceedings of 14th ACM-SIAM Symp. on Discrete Algorithms, SODA 20 03, pp. 357–363 (2003), See also `http://www.math.cmu.edu./adf/research/spectralSat`
13. Krivelevich, M., Vilenchik, D.: Solving Random Satisfiable 3CNF Formulas in Expected Polynomial Time. In: Proceedings of 17th ACM-SIAM Symp. on Discrete Algorithms (SODA 2006), pp. 454–463 (2006)
14. McSherry, F.: Spectral Partitioning of Random Graphs. In: Proceedings of 42nd Annual IEEE Symp. on Foundations of Computer Science (FOCS 2001), pp. 529–523 (2001)
15. Scott, A.D., Sorkin, G.B.: Faster Algorithms for MAX CUT and MAX CSP, with Polynomial Expected Time for Sparse Instances. In: Proceedings of APPROX-RANDOM 2003, pp. 382–395 (2003)
16. Watanabe, O., Yamamoto, M.: Average-case Analysis for the MAX-2SAT Problem. In: Biere, A., Gomes, C.P. (eds.) SAT 2006. LNCS, vol. 4121, pp. 277–282. Springer, Heidelberg (2006)

Appendix

A derivation for part (2)

The derivation for part (2) is as follows:

$$\begin{aligned}(t\boldsymbol{v}_-)^T A(t\boldsymbol{v}_-) &= t^2\beta_-^2 \boldsymbol{v}_T^{\mathbf{T}} A\boldsymbol{v}_T + 2t^2\beta_-\gamma_- \boldsymbol{v}_T^{\mathbf{T}} A\boldsymbol{v}_F + t^2\gamma_-^2 \boldsymbol{v}_F^{\mathbf{T}} A\boldsymbol{v}_F \\ &\leq \left(t^2 n\log n\right)\left((1+\epsilon)\beta_-^2 c_r + (1-\epsilon)2\beta_-\gamma_- c_p + (1+\epsilon)\gamma_-^2 c_r\right) \\ &\qquad (\because\ \beta_- > 0 \text{ and } \gamma_- < 0)\end{aligned}$$

$$
\begin{aligned}
&= \left(t^2 n \log n\right)(1+\epsilon)\left(\beta_-^2 c_r + 2\beta_-\gamma_- c_p + \gamma_-^2 c_r\right) \\
&\quad - \left(t^2 n \log n\right)(2\epsilon)(2\beta_-\gamma_- c_p) \\
&= \left(t^2 n \log n\right)\left((1+\epsilon)\left(\begin{pmatrix}\beta_- & \gamma_-\end{pmatrix}\begin{pmatrix} c_r & c_p \\ c_p & c_r\end{pmatrix}\begin{pmatrix}\beta_- \\ \gamma_-\end{pmatrix}\right) + 2\epsilon c_p\right) \\
&= \left(t^2 n \log n\right)\left((1+\epsilon)\alpha_- + 2\epsilon c_p\right) \\
&\le (t^2 n)(1-2\epsilon)\alpha_- \log n,
\end{aligned}
$$

where the last inequality comes from the fact that $c_p - c_r$ is sufficiently large.

A proof of Proposition 3

We bound it by bounding each $|\boldsymbol{v}_i^{\mathbf{T}} A \boldsymbol{v}_j|$ for $i, j \in \{T, F\}$ since

$$
|\boldsymbol{v}^{\mathbf{T}} A \boldsymbol{v}| \le \sum_{i,j \in \{T,F\}} \left|\boldsymbol{v}_i^{\mathbf{T}} A \boldsymbol{v}_j\right|.
$$

From Lemma 2, we can bound $\left|\boldsymbol{v}_i^{\mathbf{T}} A \boldsymbol{v}_j\right| \le \sqrt{c_p \log n}$ for $i \neq j$. From the remark just below Lemma 2, we can similarly bound $\left|\boldsymbol{v}_i^{\mathbf{T}} A \boldsymbol{v}_j\right| \le \sqrt{c_r \log n}$ for $i = j$. Thus, we can bound $|\boldsymbol{v}^{\mathbf{T}} A \boldsymbol{v}| \le 4\sqrt{c_p \log n}$.

A proof of Proposition 4

We first give a proof to the first inequality. We have that

$$
\begin{aligned}
\|(A - (\alpha_+ \log n)I)\boldsymbol{v}_+\|^2 &= \boldsymbol{v}_+^{\mathbf{T}}(A - (\alpha_+ \log n)I)^{\mathbf{T}}(A - (\alpha_+ \log n)I)\boldsymbol{v}_+ \\
&= \boldsymbol{v}_+^{\mathbf{T}} A^2 \boldsymbol{v}_+ - 2(\alpha_+ \log n)\boldsymbol{v}_+^{\mathbf{T}} A \boldsymbol{v}_+ + (\alpha_+ \log n)^2 \boldsymbol{v}_+^{\mathbf{T}} \boldsymbol{v}_+.
\end{aligned}
$$

We bound each term of the last formula above. For bounding the first term, we express $\boldsymbol{v}_+$ as a linear combination of orthonormal eigenvectors of A, that is, $\boldsymbol{v}_+ = \sum_{i=1}^{2n} c_i \boldsymbol{v}^{(i)}$. Note that we have that $\lambda_1 \le (1+\epsilon)\alpha_+ \log n$ **whp**. Thus, we have the following **whp**:

$$
\begin{aligned}
\boldsymbol{v}_+^{\mathbf{T}} A^2 \boldsymbol{v}_+ = \|A\boldsymbol{v}_+\|^2 &= \left\|c_1 A\boldsymbol{v}^{(1)} + \cdots + c_{2n} A\boldsymbol{v}^{(2n)}\right\|^2 \\
&= \lambda_1^2 \left\|c_1 \boldsymbol{v}^{(1)}\right\|^2 + \cdots + \lambda_{2n}^2 \left\|c_{2n} \boldsymbol{v}^{(2n)}\right\|^2 \\
&\le \lambda_1^2 \left(\|c_1 \boldsymbol{v}^{(1)}\|^2 + \cdots + \|c_{2n} \boldsymbol{v}^{(2n)}\|^2\right) \\
&\le ((1+\epsilon)\alpha_+ \log n)^2 \|\boldsymbol{v}_+\|^2.
\end{aligned}
$$

Note that we have the following **whp**:

$$
\begin{aligned}
\boldsymbol{v}_+^{\mathbf{T}} A \boldsymbol{v}_+ &\ge ((1-\epsilon)\alpha_+) n \log n \\
&= (1-\epsilon)(\alpha_+ \log n)\|\boldsymbol{v}_+\|^2.
\end{aligned}
$$

Therefore, we have the following **whp**:

$$\begin{aligned}\|(A-(\alpha_+\log n)I)\boldsymbol{v}_+\|^2 &\le ((1+\epsilon)\alpha_+\log n)^2\|\boldsymbol{v}_+\|^2\\ &\quad -2(\alpha_+\log n)(1-\epsilon)(\alpha_+\log n)\|\boldsymbol{v}_+\|^2\\ &\quad +(\alpha_+\log n)^2\|\boldsymbol{v}_+\|^2\\ &= \left((1+\epsilon)^2-2(1-\epsilon)+1\right)(\alpha_+\log n)^2\|\boldsymbol{v}_+\|^2\\ &= \left(4\epsilon+\epsilon^2\right)(\alpha_+\log n)^2\|\boldsymbol{v}_+\|^2\\ &\le (3\sqrt{\epsilon}\alpha_+\log n)^2\|\boldsymbol{v}_+\|^2.\end{aligned}$$

We next give a proof to the second inequality. In the same way as above, we have that

$$\|(A-(\alpha_-\log n)I)\boldsymbol{v}_-\|^2 = \boldsymbol{v}_-^{\mathbf{T}}A^2\boldsymbol{v}_- - 2(\alpha_-\log n)\boldsymbol{v}_-^{\mathbf{T}}A\boldsymbol{v}_- + (\alpha_-\log n)^2\boldsymbol{v}_-^{\mathbf{T}}\boldsymbol{v}_-.$$

We bound each term of the right-hand-side of the above. For bounding the first term, in the same way, we have the following **whp**:

$$\boldsymbol{v}_-^{\mathbf{T}}A^2\boldsymbol{v}_- \le ((1+\epsilon)\alpha_+\log n)^2\|\boldsymbol{v}_-\|^2.$$

For bounding the above with respect to α_-, observe that for any $\epsilon > 0$ and for any sufficiently large c_p, c_r such that $c_p - c_r$ is also sufficiently large, $|\alpha_+| \le (1+\epsilon)|\alpha_-|$. Thus, we can bound as

$$\boldsymbol{v}_-^{\mathbf{T}}A^2\boldsymbol{v}_- \le ((1+\epsilon)^2\alpha_-\log n)^2\|\boldsymbol{v}_-\|^2.$$

Furthermore, we have the following **whp**:

$$\begin{aligned}\boldsymbol{v}_-^{\mathbf{T}}A\boldsymbol{v}_- &\le (1-2\epsilon)\alpha_- n\log n\\ &= (1-2\epsilon)(\alpha_-\log n)\|\boldsymbol{v}_-\|^2.\end{aligned}$$

Therefore, we have the following **whp**:

$$\begin{aligned}\|(A-(\alpha_-\log n)I)\boldsymbol{v}_-\|^2 &\le ((1+\epsilon)^2\alpha_-\log n)^2\|\boldsymbol{v}_-\|^2\\ &\quad -2(\alpha_-\log n)(1-2\epsilon)(\alpha_-\log n)\|\boldsymbol{v}_-\|^2\\ &\quad +(\alpha_-\log n)^2\|\boldsymbol{v}_-\|^2\\ &= \left((1+\epsilon)^4-2(1-2\epsilon)+1\right)(\alpha_-\log n)^2\|\boldsymbol{v}_-\|^2\\ &\le (3\sqrt{\epsilon}\alpha_-\log n)^2\|\boldsymbol{v}_-\|^2.\end{aligned}$$

On the Expressive Power of Planar Perfect Matching and Permanents of Bounded Treewidth Matrices

Uffe Flarup[1], Pascal Koiran[2], and Laurent Lyaudet[2]

[1] Department of Mathematics and Computer Science
Syddansk Universitet, Campusvej 55, 5230 Odense M, Denmark
fax: +45 65 93 26 91
flarup@imada.sdu.dk
[2] Laboratoire de l'Informatique du Parallélisme*
Ecole Normale Supérieure de Lyon, 46, allée d'Italie, 69364 Lyon Cedex 07, France
fax: +33 4 72 72 80 80
{pascal.koiran,laurent.lyaudet}@ens-lyon.fr

Abstract. Valiant introduced some 25 years ago an algebraic model of computation along with the complexity classes VP and VNP, which can be viewed as analogues of the classical classes P and NP. Prominent examples of difficult (that is, VNP-complete) problems in this model includes the permanent and hamiltonian polynomials. In this paper we investigate the expressive power of easy special cases of these polynomials. We show that the permanent and hamiltonian polynomials for matrices of bounded treewidth both are equivalent to arithmetic formulas. Also, arithmetic weakly skew circuits are shown to be equivalent to the sum of weights of perfect matchings of planar graphs.

1 Introduction

Our focus in this paper is on easy special cases of otherwise difficult to evaluate polynomials, and their connection to various classes of arithmetic circuits. In particular we consider the permanent and hamiltonian polynomials for matrices of bounded treewidth, and sum of weights of perfect matchings for planar graphs. It is a widely believed conjecture that the permanent is hard to evaluate. Indeed, in Valiant's framework [15,16] the permanent is complete for the class VNP. This is an algebraic analogue of his $\sharp$P-completeness result for the permanent [14]. For a book-length treatment of Valiant's algebraic complexity theory one may consult [4]. The same results ($\sharp$P-completeness in the boolean framework, and VNP-completeness in the algebraic framework) also apply to the hamiltonian polynomial. The sum of weights of perfect matchings in an (undirected) graph G is yet another example of a presumably hard to compute polynomial since it reduces to the permanent when G is bipartite. However, all three polynomials are known to be easy to evaluate in special cases. In particular,

* UMR 5668 ENS Lyon, CNRS, UCBL, INRIA.

T. Tokuyama (Ed.): ISAAC 2007, LNCS 4835, pp. 124–136, 2007.

the permanent and hamiltonian polynomials can be evaluated in a polynomial number of arithmetic operations for matrices of bounded treewidth [6]. An earlier result of this flavour is Kasteleyn's theorem [7] which states that the sum of weights of perfect matchings of a planar graph can be computed in a polynomial number of arithmetic operations. One can try to turn these three efficient algorithms into general-purpose evaluation algorithms by means of reductions (this is the approach followed by Valiant in [17], where he exhibits polynomial time algorithms for several problems which previously only had exponential time algorithms, by means of holographic reductions to perfect matchings of planar graphs). For instance, in order to evaluate a polynomial P one can try to construct a matrix of bounded treewidth A such that: Entries of A are constants or variables of P, and the permanent of A is equal to P.

The same approach can be tried for the hamiltonian and the sum of weights of perfect matchings in a planar graph. The goal of this paper is to assess the power of these polynomial evaluation methods. It turns out that the three methods are all universal - that is, every polynomial can be expressed as the sum of weights of perfect matchings in a planar graph, and as a permanent and hamiltonian of matrices of bounded treewidth. From a complexity-theoretic point of view, these methods are no longer equivalent. Our main findings are that:

- The permanents and hamiltonians of matrices of polynomial size and bounded treewidth have the same expressive power, namely, the power of polynomial size arithmetic formulas. This is established in Theorem 1.
- The sum of weights of perfect matchings in polynomial size planar graphs has at least the same power as the above two representations, and in fact it is more powerful under a widely believed conjecture. Indeed, this representation has the same power as polynomial size (weakly) skew arithmetic circuits. This is established in Theorem 7. We recall that (weakly) skew arithmetic circuits capture the complexity of computing the determinant [13]. It is widely believed that the determinant cannot be expressed by polynomial size arithmetic formulas.

Our three methods therefore capture (presumably proper) subsets of the class VP of easy to compute polynomial families. By contrast, if we drop the bounded treewidth or planarity assumptions, the class VNP is captured in all three cases.

Various notions of graph "width" have been defined in the litterature besides treewidth (e.g. pathwidth, cliquewidth, rankwidth). They should be worth studying from the point of view of their expressive power. Also, Barvinok [1] has shown that if the underlying matrix has bounded rank, both the permanent and the hamiltonian polynomials can be evaluated in a polynomial number of arithmetic operations.

2 Definitions

2.1 Arithmetic Circuits

Definition 1. *An arithmetic circuit is a finite, acyclic, directed graph. Vertices have indegree 0 or 2, where those with indegree 0 are referred to as* inputs. *A*

single vertex must have outdegree 0, and is referred to as output*. Each vertex of indegree 2 must be labeled by either* $+$ *or* $\times$*, thus representing computation. Vertices are commonly referred to as* gates *and edges as* arrows*.*

By interpreting the input gates either as constants or variables it is easy to prove that each arithmetic circuit naturally represents a polynomial.

In this paper various subclasses of arithmetic circuits will be considered: For *weakly skew* circuits we have the restriction that for every multiplication gate, at least one of the incoming arrows is from a subcircuit whose only connection to the rest of the circuit is through this incoming arrow. For *skew* circuits we have the restriction that for every multiplication gate, at least one of incoming arrows is from an input gate. For *formulas* all gates (except output) have outdegree 1. Thus, reuse of partial results is not allowed. For a detailed description of various subclasses of arithmetic circuits, along with examples, we refer to [12].

Definition 2. *The* size *of a circuit is the total number of* gates *in the circuit. The* depth *of a circuit is the length of the longest path from an input gate to the output gate.*

A family (f_n) belongs to the complexity class VP if f_n can be computed by a circuit C_n of size polynomial in n, and if moreover the degree of f_n is bounded by a polynomial function of n.

2.2 Treewidth

Treewidth for undirected graphs is most commonly defined as follows:

Definition 3. *Let* $G = \langle V, E\rangle$ *be a graph. A* k*-tree-decomposition of* G *is a tree* $T = \langle V_T, E_T\rangle$ *such that:*

(i) For each $t \in V_T$ *there is a subset* $X_t \subseteq V$ *of size at most* $k+1$*.*
(ii) For each edge $(u, v) \in E$ *there is a* $t \in V_T$ *such that* $\{u, v\} \subseteq X_t$*.*
(iii) For each vertex $v \in V$ *the set* $\{t \in V_T | v \in X_t\}$ *forms a subtree of* T*.*

The treewidth of G *is the smallest* k *s.t. there exists a* k*-tree-decomposition for* G*.*

An equivalent definition is in terms of graph grammars (HR algebras [5]):

Definition 4. *A graph* G *has a* k*-tree-decomposition iff there exist a set of source labels of cardinality* $k+1$ *such that* G *can be constructed using a finite number of the following operations:*

(i) ver_a*,* $loop_a$*,* $edge_{ab}$ *(basic constructs: create a single vertex with label* a*, a single vertex with label* a *and a looping edge, two vertices labeled* a *and* b *connected by an edge)*
(ii) $ren_{a\leftrightarrow b}(G)$ *(rename all labels* a *as labels* b *and all labels* b *as labels* a*)*
(iii) $forg_a(G)$ *(forget all labels* a*)*
(iv) $G_1 \,//\, G_2$ *(composition of graphs: two vertices with same label are identified as one vertex)*

The treewidth of a directed graph is defined as the treewidth of the underlying undirected graph. The treewidth of an $(n \times n)$ matrix $M = (m_{i,j})$ is defined as the treewidth of the directed graph $G_M = \langle V_M, E_M, w \rangle$ where $V_M = \{1, \ldots, n\}$, $(i, j) \in E_M$ iff $m_{i,j} \neq 0$, and $w(i, j) = m_{i,j}$.

2.3 Permanent and Hamiltonian Polynomials

We take a graph theoretic approach to deal with permanent and hamiltonian polynomials. The reason for this being that a natural way to define the treewidth of a matrix, is by the treewidth of the underlying graph, see also e.g. [10].

Definition 5. *A* cycle cover *of a directed graph is a subset of the edges, such that these edges form disjoint, directed cycles (loops are allowed). Furthermore, each vertex in the graph must be in one (and only one) of these cycles. The* weight *of a cycle cover is the* product *of weights of all participating edges.*

Definition 6. *The* permanent *of an $(n \times n)$ matrix $M = (m_{i,j})$ is the sum of weights of all cycle covers of G_M.*

The *hamiltonian* polynomial $\mathrm{ham}(M)$ is defined similarly, except that we only sum over cycle covers consisting of a *single* cycle (hence the name).

3 Matrices of Bounded Treewidth

In this section we work with directed graphs. All paths and cycles are assumed to be directed, even if this word is omitted.

In [6] it is shown that the permanent and hamiltonian polynomials are in VP for matrices of bounded treewidth. Here we show that both the permanent and hamiltonian polynomials for matrices of bounded treewidth are equivalent to arithmetic formulas. This is an improvement on the result of [6] since the set of polynomial families representable by polynomial size arithmetic formulas is a (probably strict) subset of VP.

Theorem 1. *Let (f_n) be a family of polynomials with coefficients in a field K. The three following properties are equivalent:*

- *(f_n) can be represented by a family of polynomial size arithmetic formulas.*
- *There exists a family (M_n) of poly. size, bounded treewidth matrices such that entries of M_n are constants from K or variables of f_n, and $f_n = \mathrm{per}(M_n)$.*
- *There exists a family (M_n) of poly. size, bounded treewidth matrices such that entries of M_n are constants from K or variables of f_n, and $f_n = \mathrm{ham}(M_n)$.*

Theorem 1 follows immediately from Theorems 2, 3, 5 and 6.

Theorem 2. *Every arithmetic formula can be expressed as the permanent of a matrix of treewidth at most 2 and size at most $(n + 1) \times (n + 1)$ where n is the size of the formula. All entries in the matrix are either 0, 1, or variables.*

Proof. The first step is to construct a directed graph that is a special case of a *series-parallel* (SP) graph, in which there is a connection between weights of directed paths and the value computed by the formula. The overall idea behind the construction is quite standard, see e.g. [4] and [12]. SP graphs in general can between any two adjacent vertices have multiple directed edges. But we construct an SP graph in which there is at most one directed edge from any vertex u to any vertex v. This property will be needed in the second step, in which a connection between cycle covers and the permanent of a given matrix will be established.

SP graphs have distinguished *source* and *sink* vertices, denoted by s and t. By $SW(G)$ we denote the sum of weights of all directed paths from s to t, where the weight of a path is the *product* of weights of participating edges.

Let φ be a formula of size e. For the first step of the proof we will by induction over e construct a weighted, directed SP graph G such that $val(\varphi) = SW(G)$. For the base case $\varphi = w$ we construct vertices s and t and connect them by a directed edge from s to t with weight w.

Assume $\varphi = \varphi_1 + \varphi_2$ and let G_i be the graph associated with φ_i by the induction hypothesis. Introduce one new vertex s and let G be the union of the three graphs $\langle\{s\}\rangle$, G_1 and G_2 in which we identify t_1 with t_2 and denote it t, add an edge of weight 1 from s to s_1, and add an edge of weight 1 from s to s_2. By induction hypothesis the resulting graph G satisfies $SW(G) = 1 \cdot SW(G_1) + 1 \cdot SW(G_2) = val(\varphi_1) + val(\varphi_2)$. Between any two vertices u and v there is at most one directed edge from u to v. We introduced one new vertex, but since t_1 was identified with t_2 the number of vertices used equals $|V_1|+|V_2| \leq size(\varphi_1) + 1 + size(\varphi_2) + 1 = size(\varphi) + 1$.

Assume $\varphi = \varphi_1 * \varphi_2$. We construct G by making the disjoint union of G_1 and G_2 in which we identify t_1 with s_2, identify s_1 as s in G and identify t_2 as t in G. For every directed path from s_1 to t_1 in G_1 and for every directed path from s_2 to t_2 in G_2 we can find a directed path from s to t in G of weight equal to the product of the weights of the paths in G_1 and G_2, and since all (s,t) paths in G are of this type we get $SW(G) = SW(G_1) \cdot SW(G_2)$. The number of vertices used equals $|V_1| + |V_2| - 1 \leq size(\varphi_1) + size(\varphi_2) + 1 < size(\varphi) + 1$.

For the second step of the proof we need to construct a graph G' such that there is a relation between cycle covers in G' and directed paths from s to t in G. We construct G' by adding an edge of weight 1 from t back to s, and loops of weight 1 at all vertices different from s and t. Now, for every (s,t) path in G we can find a cycle in G' visiting the corresponding nodes. For nodes in G' not in this cycle, we include them in a cycle cover by the loops of weight 1. Because there is at most one directed edge from any vertex u to any vertex v in G' we can find a matrix M of size at most $(n+1) \times (n+1)$ such that $G_M = G'$ and $per(M) = val(\varphi)$. It can be shown that G' can be constructed using an HR algebra with only 3 source labels. □

Theorem 3. *Every arithmetic formula of size n can be expressed as the hamiltonian of a matrix of treewidth at most 6 and size at most $(2n+1) \times (2n+1)$. All entries in the matrix are either 0, 1, or variables of the formula.*

Proof. The first step is to produce the graph G as shown in Theorem 2. The next step is to show that the proof of universality for the hamiltonian polynomial in [11] can be done with treewidth at most 6. Their construction for universality of the hamiltonian polynomial introduces $|V_G| - 1$ new vertices to G in order to produce G', along with appropriate directed edges (all of weight 1). The proof is sketched in Figure 1.

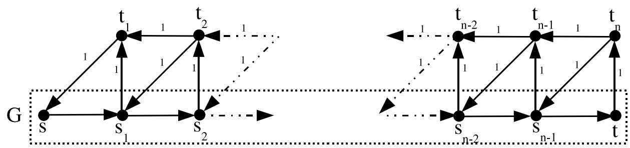


Fig. 1. Universality of the hamiltonian polynomial

The additional vertices t_i and edges permit to visit any subset of vertices of G with a directed path of weight 1 from t to s using all t_i's. Hence, any path from s to t in G can be followed by a path from t to s to obtain a hamiltonian cycle of same weight. If one just need to show universality, then it is not important exactly which one of the vertices t_i that has an edge to a given vertex among s_i. But in order to show bounded treewidth one carefully need to take into account which one of the vertices of t_i that has an edge to a particular s_i vertex. We show such a construction with bounded treewidth, by giving an HR algebra which can express a graph similar to the one in Figure 1 using 7 source labels.

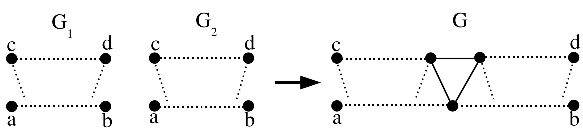


Fig. 2. Series composition (simulating multiplication)

Series composition is done using the following operations (also see Figure 2):

$$forg_e[forg_f[forg_g[ren_{d\leftrightarrow f}(ren_{b\leftrightarrow e}(G_1)) \; // \\ ren_{c\leftrightarrow g}(ren_{a\leftrightarrow e}(G_2)) \; // \; edge_{ef} \; // \; edge_{eg} \; // \; edge_{fg}]]]$$

Labels a, b, c and d in Figures 2 and 3 plays the roles of s, t, t_1 and t_n respectively in Figure 1. The above construction does not take into account, that G_1 and/or G_2 are graphs generated from the base case. For base cases vertices c and d are replaced by a single vertex. However, it is clear that the above construction can be modified to work for these simpler cases as well.

For parallel composition an additional vertex was introduced. It can be done using the following operations (also see Figure 3):

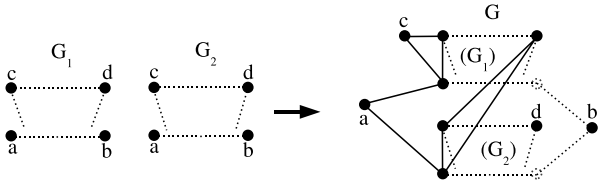


Fig. 3. Parallel composition (simulating addition)

$$
\begin{aligned}
&forg_f[ren_{a\leftrightarrow e}(ren_{c\leftrightarrow g}(\\
&\quad forg_a[forg_c[edge_{ag} \,//\, edge_{cg} \,//\, ren_{d\leftrightarrow f}(edge_{ae} \,//\, edge_{ac} \,//\, G_1)]] \,//\\
&\quad forg_a[forg_c[edge_{af} \,//\, edge_{cf} \,//\, edge_{ae} \,//\, edge_{ac} \,//\, G_2]]\\
&))]
\end{aligned}
$$

The final step in the construction, after all series and parallel composition have been done, is to connect vertices a and c and connect vertices b and d. □
By Definition 6, computing the permanent of a matrix M amounts to computing the sum of the weights of all cycle covers of G_M. In our algorithm we need to consider partial covers, which are generalizations of cycle covers.

Definition 7. *A partial cover of a directed graph is a union of paths and cycles such that every vertex of the graph belongs to at most one path (and to none of the cycles), or to at most one cycle (and to none of the paths). The weight of a partial cover is the product of the weights of all participating edges. More generally, for any set S of edges the weight $w(S)$ of S is defined as the product of the weights of the elements of S.*

In contrast to cycle covers, for a partial cover there is no requirement that all vertices be covered.

The following theorem from [2] is a standard tool in the design of parallel algorithms for graphs of bounded treewidth (see also [3] and [8]).

Theorem 4. *Let $G = \langle V, E\rangle$ be a graph of treewidth k with n vertices. Then there exists a tree-decomposition $\langle T, (X_t)_{t\in V_T}\rangle$ of G of width $3k+2$ such that $T = \langle V_T, E_T\rangle$ is a binary tree of depth at most $2\lceil log_{\frac{5}{4}}(2n)\rceil$.*

Theorem 5. *The permanent of a $n \times n$ matrix M of bounded treewidth k can be expressed as a formula of size $O(n^{O(1)})$.*

Proof. We show how to construct a circuit of depth $O(log(n))$, which can then be expressed as a formula of size $O(n^{O(1)})$. Consider the graph $G = G_M$ and apply Theorem 4 to obtain a balanced, binary tree-decomposition T of bounded width k'. For each node t of T, we denote by T_t the subtree of T rooted at t, and we denote by $X(T_t)$ the set of vertices of G which belong to X_u for at least one of the nodes u of T_t. We denote by G_t the subgraph of G induced by the subset of vertices $X(T_t)$.

Consider a partial cover C of G_t. Any given edge $(u, v) \in X_t^2$ is either used or unused by C. Likewise, any given vertex of X_t has indegree 0 or 1 in C, and outdegree 0 or 1. We denote by $\lambda_t = I_t(C)$ the list of all these data for every edge $(u, v) \in X_t^2$ and every element of X_t. By abuse of language, we will say that an edge in X_t^2 is used by λ_t if it is used by one partial cover satisfying $I_t(C) = \lambda_t$ (or equivalently, by all partial cover satisfying $I_t(C) = \lambda_t$).

We will compute for each possible list λ_t a weight $w(\lambda_t)$, defined as the sum of the weights of all partial covers C of G_t satisfying the following three properties:

(i) the two endpoints of all paths of C belong to X_t;
(ii) all uncovered vertices belong to X_t;
(iii) $I_t(C) = \lambda_t$.

Note that the number of weights to be computed at each node of T is bounded by a constant (which depends on k'). When t is the root of T we can easily compute the permanent of M from the weights $w(\lambda_t)$: it is equal to the sums of the $w(\lambda_t)$ over all λ_t which assign indegree 1 and outdegree 1 to all vertices of X_t. Also, when t is a leaf of T we can compute the weights in a constant number of arithmetic operations since G_t has at most k' vertices in this case. It therefore remains to explain how to compute the weights $w(\lambda_t)$ when t is not a leaf.

Our algorithm for this proceeds in a bottom-up manner: we will compute the weights for t from the weights already computed for its left child (denoted l) and its right child (denoted r). The idea is that we can obtain a partial cover of G_t by taking the union of a partial cover of G_l and of a partial cover of G_r, and adding some additional edges. Conversely, a partial cover of G_t induces a partial cover of G_l and a partial cover of G_r. In order to avoid counting many times the same partial cover, we must define the considered partial covers of G_l and G_r to ensure that the partial cover of G_t induces a unique partial cover of G_l and a unique partial cover of G_r. We will say that (λ_l, λ_r) is compatible with λ_t if and only if the following holds:

- no edge in X_t^2 is used in λ_l or λ_r;
- for every vertex $x \in X_t$ at most one of $\lambda_t, \lambda_l, \lambda_r$ assigns indegree 1 to x;
- for every vertex $x \in X_t$ at most one of $\lambda_t, \lambda_l, \lambda_r$ assigns outdegree 1 to x;
- every vertex $x \in X_l \backslash X_t$ has indegree 1 and outdegree 1 in λ_l;
- every vertex $x \in X_r \backslash X_t$ has indegree 1 and outdegree 1 in λ_r.

We now have to prove two things. If there is a partial cover C of G_t which satisfies the properties (i) and (ii) such that $I_t(C) = \lambda_t$ then it induces a partial cover C_l of G_l and a partial cover C_r of G_r such that C_l and C_r satisfy (i) and (ii), $I_l(C) = \lambda_l$, $I_r(C) = \lambda_r$, and (λ_l, λ_r) is compatible with λ_t. Conversely, if (λ_l, λ_r) is compatible with λ_t, and C_l and C_r are partial covers of G_l and G_r satisfying (i), (ii), $I_l(C) = \lambda_l$, and $I_r(C) = \lambda_r$, then there exists a unique partial cover C of G_t containing C_l and C_r such that $I_t(C) = \lambda_t$.

Consider a partial cover C of G_t which satisfies the properties (i) and (ii) defined above. We can assign to C a unique triple (C_l, C_r, S) defined as follows. First, we define S as the set of edges of $C \cap X_t^2$. Then we define C_l as the

set of edges of C which have their two endpoints in $X(T_l)$, and at least one of them outside of X_t. Finally, we define C_r as the set of edges of C which have their two endpoints in $X(T_r)$, and at least one of them outside of X_t. Note that $w(C) = w(C_l) \cdot w(C_r) \cdot w(S)$ since (C_l, C_r, S) forms a partition of the edges of C. Moreover, C_l is a partial cover of G_l and properties (i) and (ii) are satisfied: the endpoints of the paths of C_l and the uncovered vertices of $X(T_l)$ all belong to $X_l \cap X_t$. Likewise, C_r is a partial cover of $X(T_r)$ and properties (i) and (ii) are satisfied. If $I_l(C) = \lambda_l$ and $I_r(C) = \lambda_r$, it is clear that (λ_l, λ_r) is compatible with λ_t. Any other partition of C in three parts with one partial cover of G_l, one partial cover of G_r, and a subset of edges in X_t^2 would have an edge of X_t^2 used by C_l or C_r. Hence (λ_l, λ_r) would not be compatible with λ_t.

Suppose now (λ_l, λ_r) is compatible with λ_t, and C_l and C_r are partial covers of G_l and G_r satisfying (i), (ii), $I_l(C) = \lambda_l$, and $I_r(C) = \lambda_r$. We define S_{λ_t} as the set of edges of X_t^2 which are used by λ_t. It is clear that S_{λ_t}, C_l and C_r are disjoint. Consider $C = S_{\lambda_t} \cup C_l \cup C_r$. Since (λ_l, λ_r) is compatible with λ_t, C is a partial cover satisfying (i) and (ii). It is also clear that C is the only partial cover containing C_l and C_r such that $I_t(C) = \lambda_t$. This leads to the formula:

$$w(\lambda_t) = \sum_{(\lambda_l, \lambda_r)} w(\lambda_l) \cdot w(\lambda_r) \cdot w(S_{\lambda_t}).$$

The sum runs over all pairs (λ_l, λ_r) that are compatible with λ_t. The weight $w(\lambda_t)$ can therefore be computed in a constant number of arithmetic operations. Since the height of T is $O(log(n))$ the above algorithm can be executed on a circuit of height $O(log(n))$ as well, which then can be expressed as a polynomial size formula by duplicating subcircuits. □

Using techniques similar to that of Theorem 5 - but considering solely partial cycle covers consisting of paths - one can prove the following theorem as well:

Theorem 6. *The hamiltonian of a $n \times n$ matrix M of bounded treewidth k can be expressed as a formula of size $O(n^{O(1)})$.*

4 Perfect Matchings of Planar Graphs

Definition 8. *A* perfect matching *of a graph $G = \langle V, E \rangle$ is a subset E' of E such that every vertex in V is incident to* exactly one *edge in E'. The* weight *of a perfect matching E' is the product of weights of all edges in E'. By $SPM(G)$ we denote the sum of weights of all perfect matchings of G.*

In 1967 Kasteleyn showed in [7] that $SPM(G)$ can be computed efficiently if G is planar. His observations was that for planar graphs $SPM(G)$ could be expressed as a Pfaffian.

Theorem 7. *Let (f_n) be a family of polynomials with coefficients in a field K. The three following properties are equivalent:*

(i) (f_n) can be computed by a family of polynomial size weakly skew circuits.

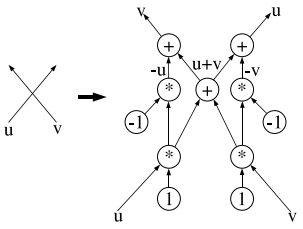


Fig. 4. Planar crossover widget for skew circuits

(ii) (f_n) *can be computed by a family of polynomial size skew circuits.*
(iii) There exists a family (G_n) *of polynomial size planar graphs with edges weighted by constants from* K *or variables of* f_n *such that* $f_n = SPM(G_n)$.

The equivalence of (i) and (ii) is etablished in [12] and [13]. In [12] the complexity class VDET is defined as the class of polynomial families computed by polynomial size (weakly) skew circuits, and it is shown that the determinant is VDET-complete. We have therefore shown that computing $SPM(G)$ for a planar graph G is equivalent to computing the determinant. Previously it was known that $SPM(G)$ could be reduced to computing Pfaffians [7]. The equivalence of (iii) with (i) and (ii) follows immediately from Theorem 8 and Theorem 9.

Theorem 8. *The output of every skew circuit of size* n *can be expressed as* $SPM(G)$ *where* G *is a weighted, planar, bipartite graph with* $O(n^2)$ *vertices. The weight of each edge of* G *is equal to 1, to -1, or to an input variable of the circuit.*

Proof. Let φ be a skew circuit; that is, for each multiplication gate *at least* one of the inputs is an input gate of φ (w.l.o.g. we assume it is *exactly* one). Furthermore, by making at most a linear amount of duplication we can assume all input gates have outdegree 1. Thus, every input gate of φ is either input to exactly one addition gate or input to exactly one multiplication gate.

Consider a drawing of φ in which all input gates which are input to an addition gate, are placed on a straight line, and all other gates are drawn on the same side of that line. Assume all arrows in the circuit are drawn as straight lines. This implies at most a quadratic number of places where two arrows cross each other.We replace crossings with the planar crossover widget from Figure 4.

For each multiplication gate we have that exactly one of the input gates is an input gate of φ, so these input gates can be placed arbitrarily close to the multiplication gate in which they are used. Thus we obtain a *planar* skew circuit φ' computing the same value as φ.

The overall idea is that every monomial in the polynomial represented by φ' will be encoded in our graph by a path-like subgraph from input s to output t. Each such subgraph has a perfect matching with weight equal to the monomial encoded on that path.

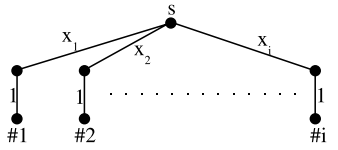


Fig. 5. Initialization for input gates which are input to addition gates

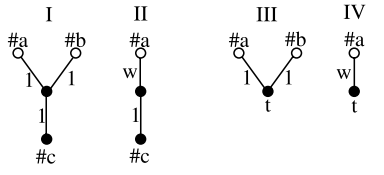


Fig. 6. I) Non-output add. II) Non-output mult. III) Output add. IV) Output mult.

Consider a topological ordering of the gates in φ' in which input gates that are input to multiplication gates have numbers less than 1, and input gates that are input to addition gates have the numbers 1 through i (where i is the number of input gates that are input to addition gates). Let m be the number of the output gate in this topological ordering of φ'. Steps 1 through i in the construction of G are shown in Figure 5. Edge weight x_i denote the input at the gate with topological number i in φ'.

For each step $i < m' \leq m$ an addition or multiplication gate is handled as shown in Figure 6. White vertices indicate vertices that are already present in the graph, whereas black vertices indicate new vertices that are introduced during that step. For simulating an addition gate we add 2 vertices and 3 edges each of weight 1. The edges are used to connect the 2 vertices that represent inputs to the addition gate. For simulating a multiplication gate we append a path of length 2 to an existing vertex. The edge weight w denote the value of the input gate of φ', which is input to that multiplication gate. Finally, the output gate of φ' is handled in a special way.

Correctness can be shown by induction using the following observation. For each step $1 \leq m' < m$ in the construction of G the following properties will hold for the graph generated so far: The labels $\sharp 1, \sharp 2, \ldots, \sharp m'$ have been assigned to m' distinct vertices. For all $1 \leq j \leq m'$ if the vertex with label $\sharp j$ is removed (along with all adjacent edges), then SPM of the remaining graph equals the value computed at gate with topological number j in φ'. □

Theorem 9. *For any weighted, planar graph G with n vertices, $SPM(G)$ can be expressed as the output of a skew circuit of size $O(n^{O(1)})$. Inputs to the skew circuit are either constants or weights of the edges of G.*

Proof. Let H be a weighted graph and $\overrightarrow{H}$ an oriented version of H. Then the Pfaffian is defined as:

$$Pf(\overrightarrow{H}) = \sum_{\mathcal{M}} sgn(\mathcal{M}) \cdot w(\mathcal{M}),$$

where $\mathcal{M}$ ranges over all perfect matchings of $\overrightarrow{H}$. The Pfaffian depends on how the edges of $\overrightarrow{H}$ are oriented, since the sign of a perfect matching depends on this orientation (details on how sign depends on orientation are not needed here).

It is known from Kasteleyn's work [7] that all planar graphs have a *Pfaffian orientation* of the edges. A Pfaffian orientation is an orientation of the edges such that each term in the above sum has positive sign $sgn(\mathcal{M})$. So for planar graphs computing $SPM(G)$ reduces to computing Pfaffians.

A Pfaffian orientation of G does not depend on the weights of the edges, it only depends on the planar layout of G. In our reduction to a skew circuit we can therefore assume that a Pfaffian orientation $\overrightarrow{G}$ is given with G, so computing $SPM(G)$ by a skew circuit is reduced to computing $Pf(\overrightarrow{G})$ by a skew circuit.

From Theorem 12 in [9] we have that $Pf(\overrightarrow{G})$ can be expressed as $SW(G')$ where G' is a weighted, acyclic, directed graph with distinguished source and sink vertices denoted s and t (recall $SW(G')$ from Theorem 2). The size of G' is polynomial in the size of $\overrightarrow{G}$.

The last step is to reduce G' to a polynomial size skew circuit representing the same polynomial. Consider a topological ordering of the vertices of G'. The vertex s is replaced by an input gate with value 1. For a vertex v of indegree 1 in G', assume u is the vertex such that there is a directed edge from u to v in G', and assume the weight of this edge is w. We then replace v by a multiplication gate, where one arrow leading to this gate comes from the subcircuit representing u, and the other arrow leading to this gate comes from a new input gate with value w. Vertices of indegree $d > 1$ are replaced by a series of $d - 1$ addition gates, adding weights of all paths leading here. □

References

1. Barvinok, A.: Two algorithmic results for the traveling salesman problem. Mathematics of Operations Research 21, 65–84 (1996)
2. Bodlaender, H.L.: NC-algorithms for graphs with small treewidth. In: van Leeuwen, J. (ed.) Graph-Theoretic Concepts in Computer Science. LNCS, vol. 344, pp. 1–10. Springer, Heidelberg (1989)
3. Bodlaender, H.L., Hagerup, T.: Parallel Algorithms with Optimal Speedup for Bounded Treewidth. In: Fülöp, Z., Gecseg, F. (eds.) Automata, Languages and Programming. LNCS, vol. 944, pp. 268–279. Springer, Heidelberg (1995)
4. Bürgisser, P.: Completeness and Reduction in Algebraic Complexity Theory. Algorithms and Computation in Mathematics, vol. 7. Springer, Heidelberg (2000)
5. Courcelle, B.: Graph Grammars, Monadic Second-Order Logic And The Theory Of Graph Minors. Contemporary Mathematics 147, 565–590 (1993)
6. Courcelle, B., Makowsky, J.A., Rotics, U.: On the fixed parameter complexity of graph enumeration problems definable in monadic second-order logic. Discrete Applied Mathematics 108, 23–52 (2001)
7. Kasteleyn, P.W.: Graph theory and crystal physics. In: Harary, F. (ed.) Graph Theory and Theoretical Physics, pp. 43–110. Academic Press, London (1967)

8. Mackworth, A.K., Zhang, Y.: Parallel and Distributed Finite Constraint Satisfaction. Technical Report 92-30, Department of Computer Science, University of British Columbia, Vancouver, B. C. Canada (1992)
9. Mahajan, M., Subramanya, P.R., Vinay, V.: The combinatorial approach yields an NC algorithm for computing Pfaffians. Discrete Applied Mathematics 143, 1–16 (2004)
10. Makowsky, J.A., Meer, K.: Polynomials of bounded treewidth. Foundations of Computational Mathematics. In: Cucker, F., Maurice Rojas, J. (eds.) Proceedings of the Smalefest 2000, pp. 211–250. World Scientific, Singapore (2002)
11. Malod, G.: Polynômes et coefficients. Ph.D. thesis (2003)
12. Malod, G., Portier, N.: Characterizing Valiant's Algebraic Complexity Classes. In: Královič, R., Urzyczyn, P. (eds.) MFCS 2006. LNCS, vol. 4162, pp. 704–716. Springer, Heidelberg (2006)
13. Toda, S.: Classes of arithmetic circuits capturing the complexity of computing the determinant. IEICE Transactions on Information and Systems E75-D, 116–124 (1992)
14. Valiant, L.G.: The complexity of computing the permanent. Theoretical Computer Science 8, 181–201 (1979)
15. Valiant, L.G.: Completeness classes in algebra. In: Proc. 11th ACM Symposium on Theory of Computing, pp. 249–261 (1979)
16. Valiant, L.G.: Reducibility by algebraic projections. In: Logic and Algorithmic (an International Symposium held in honour of Ernst Specker), pp. 365–380. Monographie n° 30 de L'Enseignement Mathématique (1982)
17. Valiant, L.G.: Holographic algorithms. In: Proc. 45th Annual IEEE Symposium on Foundations of Computer Science, pp. 306–315. IEEE Press, Los Alamitos (2004)

The 1-Versus-2 Queries Problem Revisited*

Rahul Tripathi

Department of Computer Science and Engineering, University of South Florida,
Tampa, FL 33620, USA
tripathi@cse.usf.edu

Abstract. The 1-*versus*-2 *queries* problem, which has been extensively studied in computational complexity theory, asks in its generality whether every efficient algorithm that makes at most 2 queries to a Σ_k^p-complete set L_k has an efficient simulation that makes at most 1 query to L_k. We obtain solutions to this problem under hypotheses weaker than previously considered. We prove that:

1. For each $k \geq 2$, $\mathrm{P}_{tt}^{\Sigma_k^p[2]} \subseteq \mathrm{ZPP}^{\Sigma_k^p[1]} \Longrightarrow \mathrm{PH} = \Sigma_k^p$, and
2. $\mathrm{P}_{tt}^{\mathrm{NP}[2]} \subseteq \mathrm{ZPP}^{\mathrm{NP}[1]} \Longrightarrow \mathrm{PH} = \mathrm{S}_2^p$.

Here, for a complexity class $\mathcal{C}$ and integer $j \geq 1$, we model $\mathrm{ZPP}^{\mathcal{C}[j]}$ to be the class of problems solvable by zero-error randomized algorithms that always run in polynomial time, make at most j queries to $\mathcal{C}$, and succeed with probability only $1/2 + 1/\mathrm{poly}(\cdot)$. This same model of $\mathrm{ZPP}^{\mathcal{C}[j]}$, also considered in [CC06], subsumes the class of problems solvable by randomized algorithms that always answer correctly in expected polynomial time and make at most j queries to $\mathcal{C}$.

Hemaspaandra, Hemaspaandra, and Hempel [HHH98], for $k > 2$, and Buhrman and Fortnow [BF99], for $k = 2$, had obtained the same consequence as of ours in (1) using the stronger hypothesis $\mathrm{P}_{tt}^{\Sigma_k^p[2]} \subseteq \mathrm{P}^{\Sigma_k^p[1]}$. Fortnow, Pavan, and Sengupta [FPS] had obtained the same consequence as of ours in (2) using the stronger hypothesis $\mathrm{P}_{tt}^{\mathrm{NP}[2]} \subseteq \mathrm{P}^{\mathrm{NP}[1]}$.

Our results may also be viewed as steps towards obtaining solutions to the most general form of the 1-versus-2 queries problem: For any $k \geq 1$, whether $\mathrm{P}_{tt}^{\Sigma_k^p[2]}$ can be simulated in $\mathrm{BPP}^{\Sigma_k^p[1]}$.

1 Introduction

1.1 Background

Krentel [Kre88] studied the functional version of the 1-versus-2 queries problem. Krentel proved that if every deterministic polynomial-time computable function that makes at most two queries to SAT has a deterministic polynomial-time simulation that makes at most one query to SAT, then $\mathrm{P} = \mathrm{NP}$. That is, if $\mathrm{FP}^{\mathrm{NP}[2]} \subseteq \mathrm{FP}^{\mathrm{NP}[1]}$, then $\mathrm{P} = \mathrm{NP}$.

The decision version of the 1-versus-2 queries problem has been deemed to be more difficult than its functional counterpart. A long succession of work on the decision version of the 1-versus-2 queries problem is known in the literature. We review progress made in these work.

* Research supported by the New Researcher Grant of the University of South Florida.

T. Tokuyama (Ed.): ISAAC 2007, LNCS 4835, pp. 137–147, 2007.

Kadin [Kad88] proved that if $\mathrm{P}^{\mathrm{NP}[2]} \subseteq \mathrm{P}^{\mathrm{NP}[1]}$, then $\mathrm{NP} \subseteq \mathrm{coNP/poly}$. Yap [Yap83] showed that the polynomial hierarchy collapses to the third level if $\mathrm{NP} \subseteq \mathrm{coNP/poly}$, and Cai et al. [CCHO05] improved this collapse of the polynomial hierarchy to $\mathrm{S}_2^{\mathrm{NP}}$ under the same assumption, i.e., the assumption $\mathrm{NP} \subseteq \mathrm{coNP/poly}$. Thus, Kadin's result implies that if $\mathrm{P}^{\mathrm{NP}[2]} \subseteq \mathrm{P}^{\mathrm{NP}[1]}$, then $\mathrm{PH} = \mathrm{S}_2^{\mathrm{NP}}$. Wagner [Wag89] improved Kadin's result and showed that the polynomial hierarchy collapses to $\mathrm{P}^{\Sigma_2^p}$ under the assumption $\mathrm{P}^{\mathrm{NP}[2]} \subseteq \mathrm{P}^{\mathrm{NP}[1]}$. Beigel, Chang, and Ogiwara [BCO93] built upon the work of Wagner [Wag89] and Chang and Kadin [CK96] and made further improvements to the solution of $\mathrm{P}^{\mathrm{NP}[2]} \subseteq \mathrm{P}^{\mathrm{NP}[1]}$. They showed that if $\mathrm{P}^{\mathrm{NP}[2]} \subseteq \mathrm{P}^{\mathrm{NP}[1]}$, then every set in the polynomial hierarchy can be solved by a deterministic polynomial-time Turing machine that makes at most one query to an NP oracle and at most one query to a Σ_2^p oracle. Since it is known that $\mathrm{P}^{\mathrm{NP}[2]} \subseteq \mathrm{P}^{\mathrm{NP}[1]}$ if $\mathrm{P}_{tt}^{\mathrm{NP}[2]} \subseteq \mathrm{P}^{\mathrm{NP}[1]}$ [CK95], all these results hold assuming the seemingly weaker hypothesis $\mathrm{P}_{tt}^{\mathrm{NP}[2]} \subseteq \mathrm{P}^{\mathrm{NP}[1]}$.

Hemaspaandra, Hemaspaandra, and Hempel [HHH98] studied the 1-versus-2 queries problem in a general context. They showed that for $k > 2$, if $\mathrm{P}_{tt}^{\Sigma_k^p[2]} \subseteq \mathrm{P}^{\Sigma_k^p[1]}$, then $\mathrm{PH} = \Sigma_k^p$. They extended this result for the 1-versus-2 queries problem to the result for an even more general problem: the m-versus-$(m+1)$ queries problem. In particular, they showed that for each $m > 0$ and each $k > 2$, if every deterministic polynomial-time Turing machine that makes at most $m+1$ truth-table queries to a Σ_k^p-complete set L_k has a deterministic polynomial-time simulation that makes at most m truth-table queries to L_k, then the boolean hierarchy over Σ_k^p collapses to the m'th level. That is, they showed that for each $m > 0$ and each $k > 2$, if $\mathrm{P}_{tt}^{\Sigma_k^p[m+1]} \subseteq \mathrm{P}_{tt}^{\Sigma_k^p[m]}$, then $\mathrm{DIFF}_m(\Sigma_k^p) = \mathrm{coDIFF}_m(\Sigma_k^p) = \mathrm{BH}(\Sigma_k^p)$. Beigel, Chang, and Ogiwara [BCO93] related the collapse of the boolean hierarchy to the collapse of the polynomial hierarchy. Thus, as a consequence of this relationship between the two hierarchies, the result of Hemaspaandra, Hemaspaandra, and Hempel [HHH98] also implies that for each $m > 0$ and each $k > 2$, if $\mathrm{P}_{tt}^{\Sigma_k^p[m+1]} \subseteq \mathrm{P}_{tt}^{\Sigma_k^p[m]}$, then the polynomial hierarchy can be solved by a deterministic polynomial-time Turing machine that makes $m-1$ truth-table queries to Σ_{k+1}^p and unbounded queries (strictly speaking, polynomially many queries since a deterministic polynomial-time Turing machine cannot make more than a polynomial number of queries) to Σ_k^p.

Hemaspaandra, Hemaspaandra, and Hempel [HHH98] left open the case $k = 2$ in their solution to the 1-versus-2 queries problem, which was subsequently settled by Buhrman and Fortnow [BF99]. Buhrman and Fortnow [BF99] showed that if $\mathrm{P}_{tt}^{\Sigma_2^p[2]} \subseteq \mathrm{P}^{\Sigma_2^p[1]}$, then $\mathrm{PH} = \Sigma_2^p$. They also showed that no relativizable proof technique can establish a similar result for NP. That is, they showed that the result "if $\mathrm{P}_{tt}^{\mathrm{NP}[2]} \subseteq \mathrm{P}^{\mathrm{NP}[1]}$, then $\mathrm{PH} = \mathrm{NP}$" cannot be proved using relativizable proof techniques. In spite of this negative result, they managed to obtain several other consequences, though weaker than the much sought after consequence $\mathrm{PH} = \mathrm{NP}$, of the $\mathrm{P}_{tt}^{\mathrm{NP}[2]} \subseteq \mathrm{P}^{\mathrm{NP}[1]}$ hypothesis including (a) locally either $\mathrm{NP} = \mathrm{coNP}$ or NP has polynomial-size circuits, (b) $\mathrm{PH} = \mathrm{BPP}^{\mathrm{NP}[1]}$, (c) $\Sigma_2^p \subseteq \Pi_2^p/1$, (d) $\Sigma_2^p = \mathrm{UP}^{\mathrm{NP}[1]} \cap \mathrm{RP}^{\mathrm{NP}[1]}$, and (e) $\mathrm{P}^{\mathrm{NP}} = \mathrm{P}^{\mathrm{NP}[1]}$.

In another paper, for the case $k = 2$, Hemaspaandra, Hemaspaandra, and Hempel [HHH05] extended the solution of the 1-versus-2-queries problem by Buhrman and

Fortnow [BF99] to the solution of the m-versus-$(m+1)$ queries problem. Hemaspaandra, Hemaspaandra, and Hempel showed that even for the case $k = 2$ and for all $m > 0$, if $\mathrm{P}_{tt}^{\Sigma_k^p[m+1]} \subseteq \mathrm{P}_{tt}^{\Sigma_k^p[m]}$, then $\mathrm{DIFF}_m(\Sigma_k^p) = \mathrm{coDIFF}_m(\Sigma_k^p) = \mathrm{BH}(\Sigma_k^p)$. We mention here that for the case $k = 1$ and for any $m > 1$, no solution is known for the following version of the m-versus-$(m+1)$ queries problem: whether $\mathrm{P}_{tt}^{\Sigma_k^p[m+1]}$ has a simulation in $\mathrm{P}_{tt}^{\Sigma_k^p[m]}$.

Fortnow, Pavan, and Sengupta [FPS] improved upon the result of Buhrman and Fortnow [BF99] to show that if $\mathrm{P}_{tt}^{\mathrm{NP}[2]} \subseteq \mathrm{P}^{\mathrm{NP}[1]}$, then $\mathrm{PH} = \mathrm{S}_2^p$, where S_2^p satisfies $\mathrm{S}_2^p \subseteq \mathrm{ZPP}^{\mathrm{NP}}$ ([Cai07]) $\subseteq \Sigma_2^p \cap \Pi_2^p$. This was the first solution to the 1-versus-2 queries problem for the case $k = 1$ (i.e., NP oracle) that achieved a collapse of the polynomial hierarchy to a level not above Σ_2^p. A major ingredient in their proof was a lemma, whose proof ideas were inspired from the paper of Bshouty et al. [BCG+96], that states that for every $n > 0$ and every $k > 0$, if SAT has no circuit of size n^{k+2} at length n, then there exists a polynomial-size collection S of satisfiable formulae of length n such that every circuit of size n^k fails to produce any satisfying assignment for at least one formula from S. This lemma has found applications elsewhere (see [PSV06]).

Recently, Chakaravarthy and Roy [CR06] introduced new classes O_2^p, YO_2^p, and NO_2^p as subclasses of S_2^p. These classes were introduced with the motivation of improving several existing complexity results such as the classical Karp-Lipton theorem [KL80] and a theorem of Yap [Yap83]. Chakaravarthy and Roy [CR06] showed that these new classes have implication also on the solution to the 1-versus-2-queries problem. They proved that if $\mathrm{P}_{tt}^{\mathrm{NP}[2]} \subseteq \mathrm{P}^{\mathrm{NP}[1]}$, then $\mathrm{PH} = \mathrm{NO}_2^p \cap \mathrm{YO}_2^p$. This gives a slight improvement over the solution given by Fortnow, Pavan, and Sengupta [FPS].

More recently, Chang and Purini [CP07] proved that if the NP *machine hypothesis* holds, then $\mathrm{P}_{tt}^{\mathrm{NP}[2]} \subseteq \mathrm{P}^{\mathrm{NP}[1]}$ implies $\mathrm{PH} = \mathrm{NP}$. The NP machine hypothesis postulates that there exist an $\epsilon > 0$ and a nondeterministic polynomial-time Turing machine N such that $L(N) = 0^\star$ and for any 2^{n^ϵ}-time bounded deterministic Turing machine M, $M(0^n)$ produces an accepting path of $N(0^n)$ only for finitely many n.

1.2 Our Results

In this paper, we obtain solutions to the 1-versus-2 queries problem under the hypotheses $\mathrm{P}_{tt}^{\Sigma_k^p[2]} \subseteq \mathrm{ZPP}^{\Sigma_k^p[1]}$, for integers $k \geq 1$. Note that $\mathrm{P}^{\Sigma_k^p[1]}$ is a subclass of $\mathrm{ZPP}^{\Sigma_k^p[1]}$, and so the hypotheses we consider here, namely, $\mathrm{P}_{tt}^{\Sigma_k^p[2]} \subseteq \mathrm{ZPP}^{\Sigma_k^p[1]}$, are weaker than the previously considered hypotheses (see the papers [Kad88, Wag89, BCO93, HHH98, BF99, FPS, CR06]), namely, $\mathrm{P}_{tt}^{\Sigma_k^p[2]} \subseteq \mathrm{P}^{\Sigma_k^p[1]}$. Our results may also be looked upon as steps towards obtaining solutions to the general form of the 1-versus-2 queries problem, which can be stated as whether every $\mathrm{P}_{tt}^{\Sigma_k^p[2]}$ algorithm has a $\mathrm{BPP}^{\Sigma_k^p[1]}$ simulation.

The classes $\mathrm{ZPP}^{\Sigma_k^p[1]}$, considered in the hypotheses of the statements of our results, have been recently used in some inclusion relationships between complexity classes. Cai [Cai07] proved that $\mathrm{S}_2^p \subseteq \mathrm{ZPP}^{\mathrm{NP}}$; however, the question of whether this inclusion is also an equality is open. Cai and Chakaravarthy [CC06] showed that if one restricts a ZPP algorithm so that it is allowed to make at most one query to an NP oracle, then

such an algorithm can be simulated in S_2^p. In other words, they proved that $\mathrm{ZPP}^{\mathrm{NP}[1]} \subseteq \mathrm{S}_2^p$. For $k \geq 2$, they proved that $\mathrm{ZPP}^{\Sigma_k^p[1]} \subseteq \mathrm{P}^{\Sigma_k^p[2]}$. They observed that $\mathrm{BPP} \subseteq \mathrm{ZPP}^{\mathrm{NP}[1]}$ (also proven implicitly in some other papers) and used this observation to infer that it is unlikely to prove $\mathrm{ZPP}^{\mathrm{NP}[1]} \subseteq \mathrm{P}^{\mathrm{NP}}$ unconditionally since otherwise it will yield an unconditional containment of BPP in P^{NP}, which has been open for a long time.

In all these results, the success probability of $\mathrm{ZPP}^{\mathcal{C}[1]}$ algorithms, for an arbitrary complexity class $\mathcal{C}$, is considered to be only $1/2 + 1/\mathrm{poly}(\cdot)$.

2 Preliminaries

Our alphabet is $\Sigma = \{0, 1\}$. We assume familiarity with basic notions in computational complexity theory such as complexity classes (P, NP, ZPP, BPP, Σ_k^p, Π_k^p, PH, etc.), decision problems (SAT, Σ_k^p-complete sets), oracles, reductions ($\leq_m^p$, $\leq_T^p$, $\leq_{tt}^p$), and Turing machines (deterministic, randomized). Let $\langle \cdot, \cdot \rangle$ be a multi-arity, polynomial-time computable, and polynomial-time invertible pairing function. DPTM (DPOTM) will stand for "deterministic polynomial-time (oracle) Turing machine," and RPTM (RPOTM) will stand for "randomized polynomial-time (oracle) Turing machine." For a deterministic Turing machine M and string $x \in \Sigma^*$, we use $M(x)$ to denote the computation of M on input x. Similarly, for a randomized Turing machine M and strings $x, r \in \Sigma^*$, we use $M(x; r)$ to denote the computation of M on input x and random string r. At few places, we abuse notation for the sake of brevity and let $M(x) \in \{\mathrm{Accept}, \mathrm{Reject}\}$ also denote the outcome of a deterministic Turing machine M on input x and let $M(x; r) \in \{\mathrm{Accept}, \mathrm{Reject}, ?\}$ also denote the outcome of a randomized Turing machine M on input x and random string r (which sense is being used for $M(x)$ or $M(x; r)$ will be clear from the context).

We will need the following definition:

Definition 1. *1.* [HHH98] *For any oracles C and D, let $M^{(C,D)}$ denote a* DPOTM *M making at most one query to C and at most one query to D in a truth-table fashion, i.e., in parallel. Then, for complexity classes $\mathcal{C}$ and $\mathcal{D}$,*

$$\mathrm{P}^{(\mathcal{C},\mathcal{D})} =_{df} \{L \subseteq \Sigma^* \mid (\exists C \in \mathcal{C})(\exists D \in \mathcal{D})(\exists \text{ DPOTM } M)[L = L(M^{(C,D)})]\}.$$

2. For any oracle C and integer $j \geq 0$, let $M^{C[j]}$ denote a DPOTM *M making at most j adaptive, i.e., sequential, queries to C. Then, for a complexity class $\mathcal{C}$,*

$$\mathrm{P}^{\mathcal{C}[j]} =_{df} \{L \subseteq \Sigma^* \mid (\exists C \in \mathcal{C})(\exists \text{ DPOTM } M)[L = L(M^{C[j]})]\}.$$

In this paper, we will consider zero-error randomized polynomial-time algorithms that can make at most j queries, for some $j \geq 1$, to an oracle C. The class of decision problems solvable by such algorithms is denoted by $\mathrm{ZPP}^{C[j]}$, where C is the oracle and j is a bound on the number of queries to C made by the algorithm.

Some subtlety is involved when we talk about the success probability of a bounded query randomized algorithm (e.g., a $\mathrm{ZPP}^{C[j]}$ algorithm). In particular, while the success probability of a ZPP algorithm can be amplified from $1/n^{O(1)}$ to $1/2 + 1/n^{O(1)}$ using

a standard technique, which involves running the algorithm on several independently chosen random strings and making a decision based on the outcome of the runs, the same technique does not work for the case of $\mathrm{ZPP}^{C[j]}$ algorithms since an application of the same technique could result in an algorithm that asks more than the allowed number (j) of queries to C. Hence, we need to fix our assumption regarding the success probability of $\mathrm{ZPP}^{\mathcal{C}[j]}$ algorithms (because different bounds on the success probability of $\mathrm{ZPP}^{\mathcal{C}[j]}$ algorithms could define different complexity classes).

Throughout this paper, we require the success probability of $\mathrm{ZPP}^{C[j]}$ algorithms to be only $1/2+1/\mathrm{poly}(\cdot)$. Cai and Chakaravarthy [CC06] imposed the same requirement on the success probability of $\mathrm{ZPP}^{C[j]}$ algorithms in proving their results $\mathrm{ZPP}^{\mathrm{NP}[1]} \subseteq \mathrm{S}_2^p$ and $\mathrm{ZPP}^{\Sigma_k^p[1]} \subseteq \mathrm{P}^{\Sigma_k^p[2]}$ for integers $k \geq 2$.

We next formally define $\mathrm{ZPP}^{\mathcal{C}[j]}$ for any arbitrary complexity class $\mathcal{C}$ and integer $j \geq 1$.

Definition 2 ([CC06]). *Let $\mathcal{C}$ be a complexity class and $j \geq 1$ be an integer. A set $L \in \mathrm{ZPP}^{\mathcal{C}[j]}$ if there exist a* RPOTM $M(\cdot;\cdot)$*, an oracle $C \in \mathcal{C}$, and a polynomial $p(\cdot)$ such that M satisfies the following requirements for any input $x \in \Sigma^*$:*

1. *On any random string r, $M(x;r)$ makes at most j adaptive (i.e., sequential) queries to the oracle C.*
2. *For any random string r, the output $M^{C[j]}(x;r)$ belongs to $\{\mathrm{Accept}, \mathrm{Reject}, ?\}$ such that (a) if $x \in L$, then $M^{C[j]}(x;r) \in \{\mathrm{Accept}, ?\}$ and (b) if $x \notin L$, then $M^{C[j]}(x;r) \in \{\mathrm{Reject}, ?\}$. That is, the machine M has zero-error.*
3. *M succeeds with probability at least $1/2 + 1/p(\cdot)$. That is,*

$$\mathrm{Prob}_r\left[M^{C[j]}(x;r) = \mathrm{Accept} \text{ or } M^{C[j]}(x;r) = \mathrm{Reject}\right] \geq \frac{1}{2} + \frac{1}{p(|x|)}.$$

Remarks on the robustness of the class $\mathrm{ZPP}^{\mathcal{C}[j]}$ defined in Definition 2. The class ZPP has two equivalent definitions:

1. The class of problems solvable by randomized algorithms that always answer correctly and run in expected polynomial time.
2. The class of problems solvable by zero-error randomized algorithms that always run in polynomial time and succeed with probability at least $1/\mathrm{poly}(\cdot)$ (i.e., answer correctly on at least $1/\mathrm{poly}(\cdot)$ fraction of time but may say "I don't know" on the remaining fraction of time).

These equivalences are not known to hold for $\mathrm{ZPP}^{C[j]}$. Our definition of $\mathrm{ZPP}^{C[j]}$ in Definition 2 requires the success probability to be at least $1/2 + 1/\mathrm{poly}(\cdot)$ (instead of $1/\mathrm{poly}(\cdot)$), and thus may partially capture the interpretation of $\mathrm{ZPP}^{C[j]}$ as in (2). However, we mention that our definition of $\mathrm{ZPP}^{C[j]}$ does indeed fully capture the interpretation of $\mathrm{ZPP}^{C[j]}$ as in (1), and so in this sense our definition of $\mathrm{ZPP}^{C[j]}$ is robust. To see this, let P be a problem solvable by a randomized algorithm M that always answers correctly in expected polynomial time $p(\cdot)$ and makes at most j queries to C. Consider a randomized algorithm M' that on any input x, simulates $M(x)$ for $4p(|x|)$ steps, outputs according to M if $M(x)$ halts within $4p(|x|)$ steps, and says "I don't

know" if $M(x)$ does not halt within $4p(|x|)$ steps. By Markov's inequality, it follows that M' is a j-query zero-error randomized algorithm for P, always run in time $4p(\cdot)$, and succeeds with probability at least $3/4$. Thus, P also belongs to the class $\mathrm{ZPP}^{C[j]}$ defined in Definition 2.

The class S_2^p was introduced independently by Russell and Sundaram [RS98] and by Canetti [Can96]. A formal definition of S_2^p is as follows:

Definition 3 ([Can96, RS98]). *S_2^p is the class of all sets L for which there exist a polynomial-time predicate $R(\cdot,\cdot,\cdot)$ and a polynomial $p(\cdot)$ such that for all $x \in \Sigma^*$,*

$$x \in L \Longrightarrow (\exists^p y)(\forall^p z)R(x,y,z), \text{ and}$$
$$x \notin L \Longrightarrow (\exists^p z)(\forall^p y)\neg R(x,y,z),$$

where for any w, $(\exists^p w) =_{df} (\exists w : |w| \leq p(|x|)$ and $(\forall^p w) =_{df} (\forall w : |w| \leq p(|x|))$.

For a complexity class $\mathcal{C}$, the complexity class $\mathcal{C}/\mathrm{poly}$ is defined as follows:

Definition 4 ([KL80]). *For any complexity class $\mathcal{C}$ and function $f : \mathbb{N} \to \mathbb{N}$, $\mathcal{C}/f$ denotes the class of all sets L such that for some $C \in \mathcal{C}$ and for some arbitrary function $h : \mathbb{N} \to \Sigma^*$ satisfying $(\forall n)[|h(n)| = f(n)]$, it holds that for all $x \in \Sigma^*$, $x \in L \iff \langle x, h(|x|)\rangle \in C$.*

A set L is said to be in $\mathcal{C}/\mathrm{poly}$ if and only if $L \in \mathcal{C}/p(n)$ for some polynomial p.

An important structural property of SAT is its 2-query disjunctive self-reducibility. This means that for any boolean formula $\phi(x_1, x_2, \ldots, x_n)$ of n variables, $\phi \in \mathrm{SAT}$ if and only if $\phi(x_1 := \mathit{true}, x_2, \ldots, x_n) \in \mathrm{SAT}$ or $\phi(x_1 := \mathit{false}, x_2, \ldots, x_n) \in \mathrm{SAT}$.

Due to the space limit, all proofs are omitted. They will appear in the full version of the paper.

3 Results

Hemaspaandra, Hemaspaandra, and Hempel [HHH98] showed that for any $k > 2$, if $\mathrm{P}_{tt}^{\Sigma_k^p[2]} \subseteq \mathrm{P}^{\Sigma_k^p[1]}$, then $\mathrm{PH} = \Sigma_k^p$. Buhrman and Fortnow [BF99] showed that this result extends also for the case $k = 2$: If $\mathrm{P}_{tt}^{\Sigma_2^p[2]} \subseteq \mathrm{P}^{\Sigma_2^p[1]}$, then $\mathrm{PH} = \Sigma_2^p$. In Theorem 5, we derive the same consequences under hypotheses weaker than the ones considered in the above two results.

Assuming the hypothesis $\mathrm{P}_{tt}^{\Sigma_k^p[2]} \subseteq \mathrm{ZPP}^{\Sigma_k^p[1]}$, for $k \geq 2$, we show that for a Σ_k^p-complete set L_k, its complement $\overline{L_k}$ belongs to Σ_k^p. This will prove that if $\mathrm{P}_{tt}^{\Sigma_k^p[2]} \subseteq \mathrm{ZPP}^{\Sigma_k^p[1]}$, then $\Pi_k^p \subseteq \Sigma_k^p$ and hence the polynomial hierarchy collapses to Σ_k^p.

Theorem 5. *For any integer $k \geq 2$,*

$$\mathrm{P}^{(\mathrm{NP},\Sigma_k^p)} \subseteq \mathrm{ZPP}^{\Sigma_k^p[1]} \Longrightarrow \mathrm{PH} = \Sigma_k^p.$$

We present the proof idea of Theorem 5. Fix an integer $k \geq 2$. Let L_k be a Σ_k^p-complete set. We define a set $D =_{df} L_k \times \overline{\mathrm{SAT}} \cup \overline{L_k} \times \mathrm{SAT} = \{\langle x, \psi\rangle \mid (x \in L_k \text{ and } \psi \notin \mathrm{SAT}) \text{ or } (x \notin L_k \text{ and } \psi \in \mathrm{SAT})\}$. Clearly, $D \in \mathrm{P}^{(\mathrm{NP},\Sigma_k^p)}$, and hence by the hypothesis $\mathrm{P}^{(\mathrm{NP},\Sigma_k^p)} \subseteq \mathrm{ZPP}^{\Sigma_k^p[1]}$, we have $D \in \mathrm{ZPP}^{\Sigma_k^p[1]}$ via a $\mathrm{ZPP}^{\Sigma_k^p[1]}$ machine $N^{L_k[1]}$ such that $D = L(N^{L_k[1]})$. For the sake of clarity, in the discussion that follows we omit referring to the polynomial-length bounds on quantifiers ($\exists$, $\forall$).

Because the base computation of $N^{L_k[1]}$ is a ZPP computation, we can easily describe a Σ_k^p-procedure to determine $x \notin L_k$ for an input string x, if there exist a formula ψ and a random string r such that $N^{L_k[1]}(\langle x, \psi\rangle; r)$ has a definite outcome, i.e., the outcome is either Accept or Reject but not ?, and $N^{L_k[1]}(\langle x, \psi\rangle; r)$ makes a query $\tau \in L_k$. We call such input strings x *nice*. For nice x, this Σ_k^p-procedure will include (1) a guess for the formula ψ, (2) a guess for the random string r, (3) a Σ_k^p-procedure for $\tau \in L_k$, and (4) a Σ_2^p-procedure for even k and a Π_2^p-procedure for odd k that depend on whether the outcome $N^{L_k[1]}(\langle x, \psi\rangle; r)$ is Accept or Reject, and whether ψ is in SAT or $\overline{\mathrm{SAT}}$.

The difficult part of the proof is when the input string x is not nice. In that case, we notice that for every formula ψ and for every random string r such that $N^{L_k[1]}(\langle x, \psi\rangle; r)$ reaches a definite outcome, i.e., the outcome $\in \{\mathrm{Accept}, \mathrm{Reject}\}$, the query τ made by $N^{L_k[1]}(\langle x, \psi\rangle; r)$ to L_k must be answered "no." Since the base computation of $N^{L_k[1]}$ is a ZPP computation, therefore, for every formula ψ, the fraction of random strings r for which $N^{L_k[1]}(\langle x, \psi\rangle; r)$ reaches a definite outcome is *large* (actually, this fraction is at least $1/2+1/\mathrm{poly}(|\langle x, \psi\rangle|)$). Thus, it follows that if x is not nice, then for every formula ψ, there is a large fraction of random strings r such that (1) $N^{L_k[1]}(\langle x, \psi\rangle; r)$ reaches a definite outcome $\in \{\mathrm{Accept}, \mathrm{Reject}\}$ and (2) the query τ made by $N^{L_k[1]}(\langle x, \psi\rangle; r)$ to L_k is answered "no." At this point, we define an RPTM N_{no} that on input $\langle x, \psi\rangle$, simulates $N(\langle x, \psi\rangle)$ on a uniform random string r, answers "no" to the query τ made by N, and outputs $N(\langle x, \psi\rangle; r)$. Note that N_{no} does not require oracle. We then make a crucial observation that if x is not nice, then for every ψ, N_{no} determines $\langle x, \psi\rangle \in D$ in a BPP fashion. That is, if x is not nice, then the following conditions hold for every ψ: If $\langle x, \psi\rangle \in D$, then $N_{\mathrm{no}}(\langle x, \psi\rangle)$ accepts with high probability, and if $\langle x, \psi\rangle \notin D$, then $N_{\mathrm{no}}(\langle x, \psi\rangle)$ rejects with high probability. Since a BPP computation can be performed in $\mathrm{P/poly}$, therefore, if x is not nice, then there is deterministic polynomial-time simulation of N_{no} that uses a polynomial-size advice string. (The $\mathrm{P/poly}$ simulation of N_{no} requires amplifying the success probability of N_{no}, which can be accomplished without any trouble, unlike the case of a bounded query randomized algorithm, since N_{no} does not require oracle.) Thus, it follows that if x is not nice, then for every ψ, we can determine $\langle x, \psi\rangle \in D$ in deterministic polynomial-time when given this advice string. For $k \geq 2$, we use the definition of D, the expression for $\overline{L_k}$, the $\mathrm{P/poly}$ simulation of N_{no}, and the self-reducibility of SAT to show that for a non-nice input x, there is also a Σ_k^p-procedure to determine $x \notin L_k$ (details omitted due to lack of space). We combine the two Σ_k^p-procedures (one for nice input strings and the other for non-nice input strings) to get a Σ_k^p-procedure for $\overline{L_k}$.

We mention that our idea of partitioning input strings into *nice* and *non-nice* strings is inspired from the proof of $\mathrm{ZPP}^{\mathrm{NP}[1]} \subseteq \mathrm{S}_2^p$ by Cai and Chakaravarthy [CC06]. However, we would like to stress that the results of Cai and Chakaravarthy [CC06] do not have any

direct, obvious bearings on the solutions (particularly, our solutions) to the 1-versus-2 queries problem.

We compare the proof technique of Theorem 5 with those used previously in obtaining solutions to the 1-versus-2-queries problem. The proof of the statement "if $\mathrm{P}_{tt}^{\Sigma_2^p[2]} \subseteq \mathrm{P}^{\Sigma_2^p[1]}$, then $\mathrm{PH} = \Sigma_2^p$" by Buhrman and Fortnow [BF99] used the following observation: If a set $A \in \mathrm{P}^{B[1]}$ via a DPOTM M such that $A = L(M^{B[1]})$, then there is a deterministic polynomial-time computable function $h : \Sigma^* \to \Sigma^* \times \{+, -\}$ such that $x \in A$ if and only if either $h(x) = (z, +)$ and $z \in B$ or $h(x) = (z, -)$ and $z \notin B$. Here z is the query made by $M(x)$ to B if the outcome of $M(x)$ depends on the answer to the query, and z is some fixed string known to be in (or out of) B if the outcome of $M(x)$ is independent of the answer to the query. In our proof, the query made by a $\mathrm{ZPP}^{B[1]}$ computation $M^{B[1]}$ may vary with the choice of random string r. Therefore, for our proof, we cannot say anything about the existence of a deterministic polynomial-time computable function h along the lines of the proof in [BF99]. The proof of the statement "for every integer $k > 2$, if $\mathrm{P}_{tt}^{\Sigma_k^p[2]} \subseteq \mathrm{P}^{\Sigma_k^p[1]}$, then $\mathrm{PH} = \Sigma_k^p$" by Hemaspaandra, Hemaspaandra, and Hempel [HHH98][1] used the fact that $\mathrm{P}^{\Sigma_k^p[1]}$ has $\leq_m^p$-complete sets. For our proof, we cannot use arguments similar to those in the paper [HHH98], since it is not known whether $\mathrm{ZPP}^{\Sigma_k^p[1]}$ has complete sets.

We next consider the hypothesis $\mathrm{P}_{tt}^{\mathrm{NP}[2]} \subseteq \mathrm{ZPP}^{\mathrm{NP}[1]}$. We show in Corollary 7 that the polynomial hierarchy collapses to S_2^p under this hypothesis. (Note that S_2^p is known to lie in between P^{NP} and $\Sigma_2^p \cap \Pi_2^p$.) Fortnow, Pavan, and Sengupta [FPS] obtained the same consequence under the stronger hypothesis $\mathrm{P}_{tt}^{\mathrm{NP}[2]} \subseteq \mathrm{P}^{\mathrm{NP}[1]}$. That is, they proved that if $\mathrm{P}_{tt}^{\mathrm{NP}[2]} \subseteq \mathrm{P}^{\mathrm{NP}[1]}$, then $\mathrm{PH} = \mathrm{S}_2^p$. Earlier, Buhrman and Fortnow [BF99] showed that if $\mathrm{P}_{tt}^{\mathrm{NP}[2]} \subseteq \mathrm{P}^{\mathrm{NP}[1]}$, then locally either every unsatisfiable formula has a short proof of unsatisfiability (i.e., $\mathrm{coNP} = \mathrm{NP}$) or SAT is decidable by a polynomial-size circuit. The proof of Fortnow, Pavan, and Sengupta [FPS] was built on the consequence of this result.

In Theorem 6, we derive the consequence stated in the aforementioned result of Buhrman and Fortnow [BF99] under the weaker hypothesis $\mathrm{P}_{tt}^{\mathrm{NP}[2]} \subseteq \mathrm{ZPP}^{\mathrm{NP}[1]}$. Since the proof of Fortnow, Pavan, and Sengupta [FPS] was built on this same consequence, we are able to obtain a collapse of the polynomial hierarchy to S_2^p under the hypothesis $\mathrm{P}_{tt}^{\mathrm{NP}[2]} \subseteq \mathrm{ZPP}^{\mathrm{NP}[1]}$.

Theorem 6. *If* $\mathrm{P}_{tt}^{\mathrm{NP}[2]} \subseteq \mathrm{ZPP}^{\mathrm{NP}[1]}$*, then there exist a polynomial-time predicate* R *and a constant* $k > 0$ *such that for every* n *one of the following statements is true:*

1. *Locally* $\mathrm{NP} = \mathrm{coNP}$*, i.e., for every boolean formula* ϕ *of length* n*, it holds that* $\phi \notin \mathrm{SAT} \iff (\exists w) R(\phi, w)$*, where* $|w|$ *is polynomial in* n*.*
2. *there is a circuit of size* n^k *that decides* SAT *at length* n*.*

The proof of Theorem 6 builds on a technique developed by Buhrman and Fortnow [BF99]. Their proof technique involved partitioning unsatisfiable formulas ($\overline{\mathrm{SAT}}$) into

[1] In fact, Hemaspaandra, Hemaspaandra, and Hempel [HHH98] proved a somewhat stronger statement. They proved that for any $0 \leq i < j < k$ and $i < k - 2$, if $\mathrm{P}^{(\Sigma_j^p, \Sigma_k^p)} \subseteq \mathrm{P}^{(\Sigma_i^p, \Sigma_k^p)}$, then $\mathrm{PH} = \Sigma_k^p$.

sets EASY-i and HARD-i, for each $i \in \{I, II, III, IV\}$. Then, they made use of properties of these sets in proving their aforementioned result. We modify the definitions of these sets for our purpose. We show that our new definitions of EASY-i and HARD-i sets retain some of the properties that Buhrman and Fortnow made use of in their proof. Our proof also makes use of a new notion called "P/poly-separator." This notion is a natural generalization of the notion of *polynomial-time separator*, which Buhrman and Fortnow [BF99] coined in their proof argument.

Assume that the consequent in the statement of Theorem 6 holds. Fortnow, Pavan, and Sengupta [FPS] used this assumption to show that the polynomial hierarchy collapses to S_2^p. Recently, using the same assumption, Chakaravarthy and Roy [CR06] showed that PH collapses to $\mathrm{NO}_2^p \cap \mathrm{YO}_2^p$. Thus, we get:

Corollary 7. $\mathrm{P}_{tt}^{\mathrm{NP}[2]} \subseteq \mathrm{ZPP}^{\mathrm{NP}[1]} \Longrightarrow \mathrm{PH} = \mathrm{S}_2^p = \mathrm{NO}_2^p \cap \mathrm{YO}_2^p$.

In a more recent work, Chang and Purini [CP07] showed that if the NP machine hypothesis holds, then $\mathrm{P}_{tt}^{\mathrm{NP}[2]} \subseteq \mathrm{P}^{\mathrm{NP}[1]}$ implies $\mathrm{PH} = \mathrm{NP}$. For this result, they redefined the easy and hard sets in the proof given by Buhrman and Fortnow [BF99] of the statement "$\mathrm{P}_{tt}^{\mathrm{NP}[2]} \subseteq \mathrm{P}^{\mathrm{NP}[1]}$ implies that locally either $\mathrm{NP} = \mathrm{coNP}$ or $\mathrm{NP} \subseteq \mathrm{P}/\mathrm{poly}$." Using their new definition of the easy and hard sets, Chang and Purini [CP07] were able to show that the advice in the P/poly case can be made polynomially shorter than the input length. Thus, they showed that if the P/poly case occurs infinitely often, then it would be possible to compute satisfiability in subexponential time in a way that violates the NP machine hypothesis.

We observe that the advice strings used in the proof of Theorem 6 include random strings r that can be too long for the NP machine hypothesis. Thus, it does not seem that the technique of Chang and Purini could help in showing "if the NP machine hypothesis holds, then $\mathrm{P}_{tt}^{\mathrm{NP}[2]} \subseteq \mathrm{ZPP}^{\mathrm{NP}[1]}$ implies $\mathrm{PH} = \mathrm{NP}$."

4 Conclusion and Open Problems

We obtain solutions to the 1-versus-2 queries problem under hypotheses weaker than the previously considered hypotheses, namely, $\mathrm{P}_{tt}^{\Sigma_k^p[2]} \subseteq \mathrm{P}^{\Sigma_k^p[1]}$. We show that for each $k \geq 2$, if $\mathrm{P}_{tt}^{\Sigma_k^p[2]} \subseteq \mathrm{ZPP}^{\Sigma_k^p[1]}$, then $\mathrm{PH} = \Sigma_k^p$, and if $\mathrm{P}_{tt}^{\mathrm{NP}[2]} \subseteq \mathrm{ZPP}^{\mathrm{NP}[1]}$, then $\mathrm{PH} = \mathrm{S}_2^p$. We list some open problems.

The foremost open problem is to see if our solutions can be extended to the solutions of the general 1-versus-2 queries problem: For any $k \geq 1$, whether there is a simulation of $\mathrm{P}_{tt}^{\Sigma_k^p[2]}$ in $\mathrm{BPP}^{\Sigma_k^p[1]}$. Buhrman and Fortnow [BF99] asked for implications of the hypothesis $\mathrm{BPP}^{\mathrm{NP}[2]} = \mathrm{BPP}^{\mathrm{NP}[1]}$, which is closely related to the problem we posed here. Hemaspaandra, Hemaspaandra, and Hempel [HHH98, HHH05] studied the m-versus-$(m+1)$ queries problem. They obtained solutions to this problem under the hypothesis $\mathrm{P}_{tt}^{\Sigma_k^p[m+1]} \subseteq \mathrm{P}_{tt}^{\Sigma_k^p[m]}$, for each $k \geq 2$ and each $m > 0$. It could be possible to obtain the same solutions under the weaker hypothesis $\mathrm{P}_{tt}^{\Sigma_k^p[m+1]} \subseteq \mathrm{ZPP}_{tt}^{\Sigma_k^p[m]}$, for each $k \geq 2$ and each $m \geq 2$. It is interesting to note that no solution is known for this problem under the hypothesis $\mathrm{P}_{tt}^{\Sigma_k^p[m+1]} \subseteq \mathrm{P}_{tt}^{\Sigma_k^p[m]}$, for the case $k = 1$ and any

$m \geq 2$. We would like to see a resolution of the m-versus-$(m+1)$ queries problem for this special case not only under the hypothesis $\mathrm{P}_{tt}^{\mathrm{NP}[m+1]} \subseteq \mathrm{P}_{tt}^{\mathrm{NP}[m]}$ but also under the weaker hypothesis $\mathrm{P}_{tt}^{\mathrm{NP}[m+1]} \subseteq \mathrm{ZPP}_{tt}^{\mathrm{NP}[m]}$. Finally, in this paper we require the success probability of $\mathrm{ZPP}^{\mathcal{C}[j]}$ algorithms, for any complexity class $\mathcal{C}$ and integer $j \geq 1$, to be at least $1/2 + 1/\mathrm{poly}(\cdot)$, where $\mathrm{poly}(\cdot)$ can be any arbitrary polynomial. It would be interesting to see whether our results also hold when we require the success probability of $\mathrm{ZPP}^{\mathcal{C}[j]}$ algorithms to be less than $1/2$.

References

[BCG+96] Bshouty, N., Cleve, R., Gavaldà, R., Kannan, S., Tamon, C.: Oracles and queries that are sufficient for exact learning. Journal of Computer and System Sciences 52(3), 421–433 (1996)

[BCO93] Beigel, R., Chang, R., Ogiwara, M.: A relationship between difference hierarchies and relativized polynomial hierarchies. Mathematical Systems Theory 26(3), 293–310 (1993)

[BF99] Buhrman, H., Fortnow, L.: Two queries. Journal of Computer and System Sciences 59(2), 182–194 (1999)

[Cai07] Cai, J.: S_2^p is subset of $\mathrm{ZPP}^{\mathrm{NP}}$. Journal of Computer and System Sciences 73(1), 25–35 (2007)

[Can96] Canetti, R.: More on BPP and the polynomial-time hierarchy. Information Processing Letters 57(5), 237–241 (1996)

[CC06] Cai, J., Chakaravarthy, V.: On zero error algorithms having oracle access to one query. Journal of Combinatorial Optimization 11(2), 189–202 (2006)

[CCHO05] Cai, J., Chakaravarthy, V., Hemaspaandra, L., Ogihara, M.: Competing provers yield improved Karp-Lipton collapse results. Information and Computation 198(1), 1–23 (2005)

[CK95] Chang, R., Kadin, J.: On computing boolean connectives of characteristic functions. Mathematical Systems Theory 28(3), 173–198 (1995)

[CK96] Chang, R., Kadin, J.: The boolean hierarchy and the polynomial hierarchy: A closer connection. SIAM Journal on Computing 25(2), 340–354 (1996)

[CP07] Chang, R., Purini, S.: Bounded queries and the NP machine hypothesis. In: Proceedings of the 22nd Annual IEEE Conference on Computational Complexity, pp. 52–59. IEEE Computer Society Press, June (2007)

[CR06] Chakaravarthy, V., Roy, S.: Oblivious symmetric alternation. In: Durand, B., Thomas, W. (eds.) STACS 2006. LNCS, vol. 3884, pp. 230–241. Springer, Heidelberg (2006)

[FPS] Fortnow, L., Pavan, A., Sengupta, S.: Proving SAT does not have small circuits with an application to the two queries problem. Journal of Computer and System Sciences (to appear)

[HHH98] Hemaspaandra, E., Hemaspaandra, L., Hempel, H.: A downward collapse within the polynomial hierarchy. SIAM Journal on Computing 28(2), 383–393 (1998)

[HHH05] Hemaspaandra, E., Hemaspaandra, L., Hempel, H.: Extending downward collapse from 1-versus-2 queries to m-versus-$m+1$ queries. SIAM Journal on Computing 34(6), 1352–1369 (2005)

[Kad88] Kadin, J.: The polynomial time hierarchy collapses if the boolean hierarchy collapses. SIAM Journal on Computing 17(6), 1263–1282 (1988) Erratum appears in the same journal 20(2) 404

[KL80] Karp, R., Lipton, R.: Some connections between nonuniform and uniform complexity classes. In: Proceedings of the 12th ACM Symposium on Theory of Computing, pp. 302–309. ACM Press, New York (1980)

[Kre88] Krentel, M.: The complexity of optimization problems. Journal of Computer and System Sciences 36(3), 490–509 (1988)

[PSV06] Pavan, A., Santhanam, R., Vinodchandran, N.: Some results on average-case hardness within the polynomial hierarchy. In: Proceedings of the 26th Conference on Foundations of Software Technology and Theoretical Computer Science, pp. 188–199 (2006)

[RS98] Russell, A., Sundaram, R.: Symmetric alternation captures BPP. Computational Complexity 7(2), 152–162 (1998)

[Wag89] Wagner, K.: Number-of-query hierarchies. Technical Report 4, Institut für Informatik, Universität Würzburg, Würzburg, Germany (February 1989)

[Yap83] Yap, C.: Some consequences of non-uniform conditions on uniform classes. Theoretical Computer Science 26, 287–300 (1983)

Approximating the Crossing Number of Toroidal Graphs

Petr Hliněný[1,2,⋆] and Gelasio Salazar[3,⋆⋆]

[1] FEI, Technical University of Ostrava
17. listopadu 15, 708 33 Ostrava, Czech Republic
[2] Faculty of Informatics, Masaryk University
Botanická 68a, 602 00 Brno, Czech Republic
hlineny@fi.muni.cz
[3] Instituto de Física, Universidad Autónoma de San Luis Potosí
San Luis Potosí, Mexico, 78000
gsalazar@ifisica.uaslp.mx

Abstract. CrossingNumber is one of the most challenging algorithmic problems in topological graph theory, with applications to graph drawing and VLSI layout. No polynomial time constant approximation algorithm is known for this NP-complete problem. We prove that a natural approach to planar drawing of toroidal graphs (used already by Pach and Tóth in [21]) gives a polynomial time constant approximation algorithm for the crossing number of toroidal graphs with bounded degree. In this proof we present a new "grid" theorem on toroidal graphs.

Keywords: crossing number, approximation algorithm, toroidal graph, edge-width, toroidal grid.
2000 Math Subject Classification: **05C10, 05C62, 68R10**

1 Introduction

We assume the reader is familiar with the standard terminology of graph theory. In this paper we consider finite graphs, with loops or multiple edges allowed. Some standard topological graph theory terminology is briefly introduced throughout this paper. For other related terminology and theory we refer the reader to Mohar and Thomassen [19]. Here our main interest lies in *toroidal graphs*, that is, graphs that can be *embedded* (meaning drawn without edge crossings) on the torus.

The (planar) *crossing number* $\mathrm{cr}(G)$ of a graph G is the minimum number of edge crossings in a drawing of G in the plane. To resolve ambiguity, we consider drawings of graphs such that no edge passes through another vertex, and that no three edges intersect in a common point which is not a vertex. Then a *crossing* is an intersection point of two edges which is not a vertex.

⋆ Supported by Czech research grant GAČR 201/05/0050 and by the program "Information Society" of the Czech Academy of Sciences, project No. 1ET101940420.
⋆⋆ Supported by CONACYT Grant 45903.

T. Tokuyama (Ed.): ISAAC 2007, LNCS 4835, pp. 148–159, 2007.

Computing crossing numbers has important applications in VLSI design, and, naturally, in the graph drawing area. The algorithmic decision problem of *crossing minimization* is formulated as follows:

CrossingNumber

Input: A (multi)graph G and an integer k.
Question: Is $\mathrm{cr}(G) \leq k$? (Possibly: if so, find the corresponding drawing).

The problem is in NP since one could guess the optimal drawing, replace its crossings with new (degree 4, subdividing) vertices, and verify planarity of the resulting graph. It has been proved by Garey and Johnson [9] that crossing minimization is NP-complete if k is a part of the input. The same assertion has been proved true later by Hliněný [12] both for cubic graphs and for the minor-monotone version (cf. [1]) of crossing number. An important, stubborn open problem is to decide whether the crossing number of graphs with bounded tree-width can be computed in polynomial time.

On the positive side, a surprising result from Grohe [11], recently improved by Kawarabayashi and Reed [15], states that CrossingNumber is an FPT problem. Unfortunately, these algorithms are not usable in practice, not even for small values of k. Regarding approximability results, the best general result known to date is a polynomial time algorithm by Even, Guha and Schieber [8], which approximates $\mathrm{cr}(G) + |V(G)|$ up to a factor of $\log^3 |V(G)|$ for graphs G of bounded degree (notice the $+|V(G)|$ term).

Our interest in the crossing number of graphs embedded in a given surface follows a recent major trend in crossing numbers research, which emphasizes the relationship of crossing number to topological graph theory and to structural parameters (see for instance [1,2,3,10,15,23]). Böröczky, Pach and Tóth [3,21] prove that the crossing number of a toroidal graph G is at most $c \cdot \Delta(G)|V(G)|$, with an analogous generalization to any fixed surface. A refinement of this estimate bounds $\mathrm{cr}(G)$ by a factor of the sum of square degrees of G. In this direction the asymptotically best possible estimate for graphs G of orientable genus $g = o(|V(G)|)$ is $\mathrm{cr}(G) \leq c \cdot g\Delta(G)|V(G)|$ given by Djidjev and Vrt'o [6]. An even wider generalization of the problem by Telle and Wood [23] shows that any class $\mathcal{G}$ of bounded-degree graphs excluding a fixed minor H satisfies $\mathrm{cr}(G) \leq c_{H,\Delta} \cdot |V(G)|$ for every $G \in \mathcal{G}$. Although all these estimates are tight in the sense that there exist graph sequences attaining them asymptotically, they give no good algorithmic approximation for CrossingNumber since many other graphs in these classes also have arbitrarily smaller crossing number.

On the other hand, constant factor approximation algorithms of CrossingNumber are known only for some particular families of graphs, such as [10] for projective graphs of bounded degree with an approximation factor $4.5\Delta(G)^2$; or [13] for almost planar graphs of bounded degree with an approximation factor $\Delta(G)$. A graph is *almost planar* if deleting one edge leaves it planar. In this relation one should mention that an older result of Riskin [22] implies that, for almost planar graphs coming from cubic 3-connected planar subgraphs, the crossing number can be determined exactly. Other structural aspects of the crossing

number of almost planar graphs are dealt with by Mohar [18]. Now we extend our attention to graphs embeddable on the torus.

The new contribution of our paper lies in a fine analysis of a natural planar-drawing algorithm for toroidal graphs (analogous to the approach of Pach and Tóth [21]), which is complemented with a matching lower bound on the crossing number. This is summarized next. We refer to Section 2 for the definition of edge-width, and to Lemma 3.2 for details on the $o(1)$ term appearing there.

Theorem 1.1. *Given a nonplanar toroidal graph G, one can construct in polynomial $O(n \log n)$ time, where $n = |V(G)| + |E(G)|$, a drawing of G in the plane*

a) with at most $\big(4.5\Delta(G)^2 + o(1)\big) \cdot \mathrm{cr}(G)$ crossings;

b) with at most $6\Delta(G)^2 \cdot \mathrm{cr}(G)$ crossings if G embeds in the torus with dual edge-width at least $8\Delta(G)$.

Hence for a fixed maximal degree bound $\Delta(G) \leq \Delta$ we get (b) a polynomial time algorithm which approximates CrossingNumber up to a constant factor $6\Delta^2$ for all graphs which have sufficiently "dense" toroidal embeddings. Notice that, concerning time complexity of our algorithm, we may assume $n = |V(G)|$ if $\Delta(G)$ is bounded, or if G is simple.

Our paper is organized as follows. In Section 2 we describe Drawing Algorithm 2.3 (cf. Theorem 1.1) and some details of its implementation. It uses a natural idea of surgery along a manifold, extensively used in classical topology: "cut and open" a toroidal embedding of a given graph G along a curve intersecting the fewest number of edges, and then redraw the affected edges of G inside the rest of the embedding in the best possible (crossing-wise) way. We prove in Section 3 that this approach gives a good approximation of the correct crossing number of G by exhibiting in G a special minor (a toroidal grid) which itself has crossing number very close to the quantity computed in Algorithm 2.3. This part represents the main new contribution of our paper, not appearing in any of the related previous papers [3,21,23,2]. Theoretical details about finding this grid minor are then given in Section 4.

2 The Algorithm

For the coming arguments we have to introduce some common topological terms. A closed curve on a surface is simply called a *loop*. Two loops α, β on a surface Σ are *freely homotopic* if α can be continuously transformed to β on Σ. A closed curve on a surface is *contractible* if it is freely homotopic to a constant curve (it can be continuously deformed to a single point).

Since we are going to work with a toroidal embedding of a given graph, we first resolve the task of finding it. It is widely known how to test planarity efficiently, and a strong generalization of that result by Mohar [17] claims:

Theorem 2.1 (Mohar). *For every surface Σ there is a linear time algorithm which, for a given graph G, either finds an embedding of G on Σ or returns a subgraph of G that is a subdivision of a "minimal obstacle" for Σ.*

In particular, this result provides us a toroidal embedding of the input graph which is known to be toroidal.

The second ingredient in our approach is a well-known concept of measuring "dual density" of a graph embedding. Consider now a graph G embedded on a nonplanar surface Σ (i.e. G is a topological rather than a combinatorial object, and the embedding G itself determines the surface Σ). The *edge-width* $\mathrm{ew}(G)$ of the embedding G is then defined as the length of the shortest cycle in G which is not contractible on Σ.

The edge-width of a given embedding is efficiently computable [19, 4.2]. Faster recent algorithms appear, e.g., in [5] or [16].

Theorem 2.2 (Kutz). *Given an embedded graph H on an orientable surface, one can compute in time $O(n \log n)$, where $n = |V(H)| + |E(H)|$, the edge-width k of the embedding H, and find a length-k noncontractible cycle in H.*

The basic idea—"cut and open" a toroidal embedding of a given graph G while affecting the fewest number of edges, appears in the core of the proof by Pach and Tóth [21] (Böröczky et al [3]). We adopt it (with a slight modification – using the topological dual instead of a triangulation) in an algorithmical setting. See Fig. 1 for an informal hint to geometric idea of this algorithm.

Algorithm 2.3. Drawing a toroidal graph G in the plane.

1. Given a toroidal graph G, we first test planarity of G. (If G is plane, we are done.) We construct an embedding $\bar{G}$ of G on the torus $\mathcal{S}_1$ using Theorem 2.1.
2. We construct the topological dual G^* for $\bar{G}$ on $\mathcal{S}_1$. We compute k, the edge-width of G^*, and the corresponding length-k cycle C^* in G^* as described in Theorem 2.2.
3. Let γ be the simple loop of $\mathcal{S}_1$ formed by C^*. We transform $\mathcal{S}_1$ into a cylinder $\mathcal{R}$ by "cutting along" γ. The cylinder $\mathcal{R}$ has two boundary curves γ_1 and γ_2 which are the copies of γ. In this way the embedded (dual) graph G^* is naturally transformed into $G^\star$ on $\mathcal{R}$ such that $C_1^\star$ and $C_2^\star$ are the two copies of C^* embedded as γ_1 and γ_2, respectively.
4. Let G^o be the graph resulting from $G^\star$ by contracting each of $C_1^\star$ and $C_2^\star$ into single vertices w_1 and w_2. Note that since G is not planar, it follows that G^o is connected. We then use breadth-first search to compute the shortest path P^o of length ℓ between w_1 and w_2 in G^o. Let δ be the simple curve on $\mathcal{R}$ formed by the embedding of P^o in $G^\star$. Hence δ connects a point x_1 on γ_1 to a point x_2 on γ_2, and δ intersects ℓ edges of the original embedding $\bar{G}$.
5. Let $F \subseteq E(G)$ be the set of those edges in the embedding $\bar{G}$ which are crossed by γ, and $F' \subseteq E(G)$ be the set of those crossed by δ. Hence $\bar{G} - F$ is actually embedded on $\mathcal{R}$, and we extend this crossing-free subdrawing of $G - F$ into a new drawing $\widetilde{G}$ of the whole graph G on $\mathcal{R}$ as follows: each edge from F is newly drawn along an appropriate section of γ_1 up to x_1, then along δ (crossing the ℓ edges from F') until reaching x_2, and finally along an appropriate section of γ_2. We output $\widetilde{G}$ as a drawing of G.

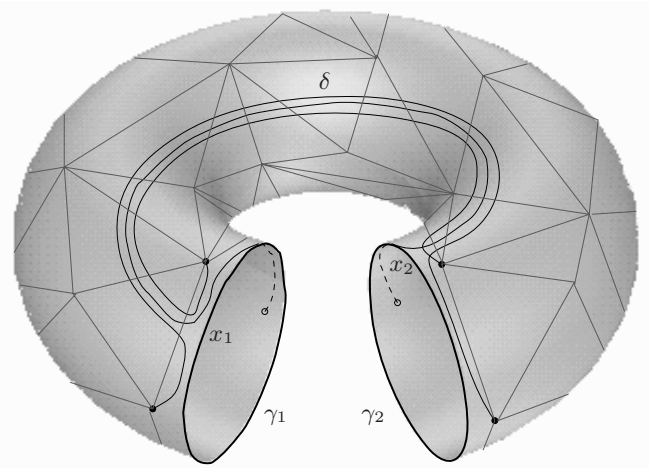


Fig. 1. Cutting a toroidal embedding by γ, and redrawing the affected edges along δ

Lemma 2.4. *The output graph $\widetilde{G}$ in Algorithm 2.3 is a planar drawing of G with at most $k\ell + \lfloor k^2/4 \rfloor$ edge crossings, where k, ℓ are computed in the algorithm.*

Proof. Since cutting the torus along any noncontractible loop (Step 3) results in a cylinder, the graph $\bar{G} - F$ is cylinder-embedded, and hence plane (F are the edges intersected by γ, that is, dual to $E(C^*)$). Now since G is nonplanar, $k > 0$ and the graph G^o is connected. So in Step 4 we find a dual path P^o and the associated curve δ connecting x_1 with x_2 on the two boundaries γ_1, γ_2 of our cylinder $\mathcal{R}$.

The drawing $\bar{G} - F$ is disjoint from both γ_1, γ_2 in $\mathcal{R}$, and by the definition of F, each $e = u_1u_2 \in F$ has u_i on the face incident with γ_i, $i = 1, 2$. Hence such $e \in F$ can be drawn along γ_i from u_i to x_i without crossings, for $i = 1, 2$, and (in the middle) along δ making ℓ crossings with the edges from F'. See in Fig. 1. Furthermore, two edges $e, e' \in F$ must cross each other in $\widetilde{G}$ if and only if x_1, x_2 (visualized as points back on γ) separate the intersections $e \cap \gamma$ from $e' \cap \gamma$. This makes at most $\lfloor k/2 \rfloor \lceil k/2 \rceil = \lfloor k^2/4 \rfloor$ crossings in addition to the $k\ell$ crossings between F and F'. ∎

Lemma 2.5. *Algorithm 2.3 runs in time $O(n \log n)$ where $n = |V(G)| + |E(G)|$.*

Proof. We represent an embedded graph by its rotation system (of edges at the vertices). Step 1 runs in linear time with this representation, by Theorem 2.1. Now the dual embedding G^* is easily obtained in linear time, too, and so Step 2 runs in time $O(n \log n)$, by Theorem 2.2. The transformation into a cylindrical embedding $G^\star$ described in Step 3 is simply done in $O(k)$ time: we duplicate C^* into $C_1^\star, C_2^\star$ and "split" the local rotations of $V(C^*)$ accordingly. In Step 4 we deal with an abstract graph G^o, and the breadth-first search (for P^o) on it also runs in $O(n)$ time. Then in Step 5 we get the embedding $\bar{G} - F$ in linear time

as the plane dual of $G^\star$, excluding the $C_1^\star, C_2^\star$-faces. We also identify F' as the edge set dual to $E(P^o)$ computed in Step 4.

Finally, we visualize each crossing in the final planar drawing $\widetilde{G}$ as a ("dummy") degree 4 vertex in an associated planar graph G'. Knowing C^* and P^o, and their dually associated edge sets F and F', the construction of G' is computationally achieved in time $O(k\ell+k^2)$ (see Lemma 2.4), which is also $O(n)$ by further Lemma 3.1 and Theorem 3.3. ∎

3 Lower Crossing Bound

Let G be a graph with maximum degree $\Delta(G) = \Delta$. For simplification we consider G already *embedded in the torus*. As we have observed in Lemma 2.4, Algorithm 2.3 yields a drawing of G in the plane with at most $k\ell+k^2/4$ crossings. Our basic claim, in order to prove Theorem 1.1 that this drawing is a good estimate for $\mathrm{cr}(G)$, is that the computed quantity $k\ell$ is within a constant factor of $\mathrm{cr}(G)$ (more precisely, a factor that depends only on Δ for large enough k). Equivalently, there is a suitable function $f(\Delta) > 0$ such that $\mathrm{cr}(G) \geq (f(\Delta) - o_k(1)) \cdot k\ell$. The goal of this section is to prove this claim.

Assuming a cycle C and a path P with both ends on C, simultaneously embedded in an orientable surface Σ, we say that P is *C-separated* if P is internally disjoint from C, and the first and the last edges of P appear on opposite sides of the loop C in Σ. To give formal mathematical meaning of the variables k, ℓ in Algorithm 2.3, we let $\mathrm{ew}^*(G)$ denote the *dual edge-width* (the edge-width of the topological dual of G), and we let $\mathcal{L}(G)$ denote the set of *orthogonal widths*, that is, the set of all integers ℓ possessing the following property: there is a noncontractible cycle C^* of length $\mathrm{ew}^*(G)$ in the topological dual G^*, such that ℓ is the length of the shortest path P^* in G^* with both ends in $V(C^*)$ which is C^*-separated. Note that P^* may be a cycle, and so with a slight abuse of terminology we do allow the ends of P^* to be the same.

Clearly, Algorithm 2.3 computes $k = \mathrm{ew}^*(G)$ and $\ell \in \mathcal{L}(G)$.

Lemma 3.1. *If $k = \mathrm{ew}^*(G)$ and $\ell \in \mathcal{L}(G)$, then $\ell \geq k/2$.*

Proof. Let C^* be a dual cycle of G of length k, and P^* be path of length ℓ as above. Seeking a contradiction, we suppose that $\ell < k/2$. The ends of P^* on C^* determine two subpaths of C^* (both with the same ends as P^*), and one of them, say Q^*, has length at most $k/2$. Then $Q^* \cup P^*$ is noncontractible and its length is at most $\ell + k/2 < k/2 + k/2 = \mathrm{ew}^*(G)$, a contradiction. ∎

The main step towards Theorem 1.1 follows now.

Lemma 3.2. *Let G be a graph embedded in the torus with maximum degree Δ, $k = \mathrm{ew}^*(G)$ and $\ell = \max \mathcal{L}(G)$. Then*

$$\mathrm{cr}(G) \;\geq\; \left(\frac{1}{12\lfloor \Delta/2 \rfloor^2} - o_k(1)\right) \cdot k\ell \;\geq\; \left(\frac{1}{3\Delta^2} - o_k(1)\right) \cdot k\ell\,,$$

where $o_k(1) \to 0$ as $k \to \infty$ with fixed Δ.

Before we move on with the proof, we recall that the $p \times q$ *toroidal grid* is the Cartesian product $C_p \times C_q$ of cycles of lengths p and q. This 4–regular graph embeds naturally in the torus with the edge-width $\min\{p, q\}$.

Proof. The main new ingredient for the proof is the following statement, which guarantees the existence of a large toroidal grid minor contained in G.

Theorem 3.3. *Let G be a graph embedded in the torus, $k = \mathrm{ew}^*(G)$ and $\ell \in \mathcal{L}(G)$. Then G contains a minor isomorphic to the toroidal grid of size*

$$\max\left(\left\lfloor \frac{2}{3}\frac{k}{\lfloor \Delta/2\rfloor}\right\rfloor, \left\lceil \frac{\ell}{\lfloor \Delta/2\rfloor}\right\rceil\right) \times \left\lfloor \frac{2}{3}\frac{k}{\lfloor \Delta/2\rfloor}\right\rfloor .$$

Assuming this result for the moment (we devote the next section to its proof), we finish the proof of Lemma 3.2.

First we recall that if H is a minor of G, and H has maximum degree at most 4, then $\mathrm{cr}(G) \geq \frac{1}{4}\,\mathrm{cr}(H)$ [20]. It is known that the crossing number of the toroidal grid of size $p \times q$, where $p \geq q \geq 3$, is at least $\frac{1}{2}(q-2)p$ by [14]. Combining these facts with Theorem 3.3, we obtain

$$\mathrm{cr}(G) \geq \frac{1}{4}\cdot\frac{1}{2}(q-2)p \geq \frac{1}{8}\cdot\frac{2k}{3\lfloor \Delta/2\rfloor}\cdot\frac{\ell}{\lfloor \Delta/2\rfloor} - O(\ell) \geq \left(\frac{1}{12\lfloor \Delta/2\rfloor^2} - o_k(1)\right)\cdot k\ell\,.$$

∎

To derive Theorem 1.1(a) from Lemmas 2.4, 2.5 and this estimate of Lemma 3.2, we note that, using Lemma 3.1 in the first step,

$$k\ell + k^2/4 \leq k\ell + k\ell/2 \leq \mathrm{cr}(G)\cdot\frac{3}{2}\left(\frac{1}{3\Delta^2} - o_k(1)\right)^{-1} \leq \mathrm{cr}(G)\cdot\left(\frac{9}{2}\Delta^2 + o_k(1)\right).$$

The same argument proves also part b) of Theorem 1.1, with a constant factor $\frac{3}{2}\,4\Delta^2 = 6\Delta^2$, if we adapt Lemma 3.2 without asymptotic terms:

Corollary 3.4. *Let G be a graph embedded in the torus, $k = \mathrm{ew}^*(G)$ and $\ell = \max\mathcal{L}(G)$. If $k \geq 16\lfloor \Delta/2\rfloor$, then $\mathrm{cr}(G) \geq \frac{1}{4\Delta^2}\cdot k\ell$.*

Proof. We just slightly modify the last line of the proof of Lemma 3.2:

$$\mathrm{cr}(G) \geq \frac{1}{8}\left(\frac{2k}{3\lfloor \Delta/2\rfloor} - \frac{2}{3} - 2\right)\cdot\frac{\ell}{\lfloor \Delta/2\rfloor} \geq \frac{1}{8}\cdot\frac{k}{2\lfloor \Delta/2\rfloor}\cdot\frac{\ell}{\lfloor \Delta/2\rfloor} \geq \frac{1}{4\Delta^2}\cdot k\ell\,.$$

∎

4 Finding a Grid Minor

For readers' convenience, we use throughout the coming arguments the same notation as introduced in Algorithm 2.3 and used in Theorem 3.3.

Thus, let G be a graph embedded in the torus $\mathcal{S}_1$ with maximum degree Δ, and G^* be its topological dual. Although its embedding may not be unique, the following arguments can use *any embedding* G in $\mathcal{S}_1$ to derive the conclusions.

Set $k = \mathrm{ew}^*(G)$ and choose *any* orthogonal width (see in Section 3) $\ell \in \mathcal{L}(G)$. Consequently select any appropriate C^*, a length-k noncontractible cycle in G^* such that the shortest C^*-separated path P^* in G^* has length ℓ, and denote by γ the simple loop in $\mathcal{S}_1$ determined by C^*.

Recall, in order to finish the arguments of Section 3 we have to provide a proof of Theorem 3.3, that is, find a sufficiently large toroidal grid minor in the graph G relatively to the parameters k, ℓ. The *face-width* $\mathrm{fw}(G)$ of an embedded graph G is the smallest number of points in which a noncontractible loop on the surface intersects G. It is strongly related to dual edge-width as $\mathrm{ew}^*(G) \leq \lfloor \Delta(G)/2 \rfloor \cdot \mathrm{fw}(G)$. A nice result by de Graaf and Schrijver [7] says:

Theorem 4.1 (de Graaf and Schrijver). *Any graph embedded on the torus contains a minor isomorphic to the $s \times s$–toroidal grid if $s = \lfloor 2\,\mathrm{fw}(G)/3 \rfloor \geq 5$.*

This theorem is, unfortunately, not directly usable in our context since the lower bound on the size of a toroidal grid (cf. Theorem 3.3) implied by it would be of order $k \times k$, and not of $k \times \ell$ as we need. Hence we need a generalization of it respecting our "two-directional" parametrization in k, ℓ. We, however, do not see (whether and) how to generalize the methods of de Graaf and Schrijver [7]—their proof relies on difficult results within geometry of numbers. That is why we turn to a graph-theoretical alternative here.

We remark that, although it would likely be possible to derive existence of a $\Omega(k) \times \Omega(\ell)$ toroidal grid minor in our G from the details of the proofs in [4, Section 8], that approach would not be immediate either, and our chosen elementary way seems to provide better constants after all.

We start with the following two, similarly looking, claims. The first one has a straightforward elementary proof using Menger's theorem, while the second one simply takes one of the two cycle families from Theorem 4.1.

Lemma 4.2. *Let G, γ and k, ℓ be as above. Then the embedded graph G contains*

a) at least $\left\lceil \frac{\ell}{\lfloor \Delta/2 \rfloor} \right\rceil$ pairwise disjoint cycles, all freely homotopic to γ.

b) at least $\left\lfloor \frac{2}{3} \frac{k}{\lfloor \Delta/2 \rfloor} \right\rfloor$ pairwise disjoint pairwise freely homotopic cycles which are not homotopic to an iteration of γ.

We skip the easy proofs here, and we move onto the main theorem.

Proof of Theorem 3.3. Let our graph G be embedded in the torus $\mathcal{S}_1$. For $p = \left\lceil \frac{\ell}{\lfloor \Delta/2 \rfloor} \right\rceil$ and $q = \left\lfloor \frac{2}{3} \frac{k}{\lfloor \Delta/2 \rfloor} \right\rfloor$, we denote by $C_1, C_2, \ldots, C_p$ the pairwise disjoint cycles in G by Lemma 4.2(a) and by $D_1, D_2, \ldots, D_q$ those cycles by Lemma 4.2(b). If $p < q$, then our statement is a consequence of Theorem 4.1. So we may assume $p \geq q$, and $p \geq 3$ since for $p \leq 2$ the statement is then trivial. To simplify notation, we use *cyclic indexing* of the C-cycles modulo p and of the D-cycles modulo q. We also let $C_+ := C_1 \cup C_2 \cup \ldots \cup C_p$ and $D_+ := D_1 \cup D_2 \cup \ldots \cup D_q$.

Remark. It may appear that we already have the desired grid as a minor in $C_+ \cup D_+$, since every D_j, $j \in \{1, \ldots, q\}$, has to intersect each C_i, $i \in \{1, \ldots, p\}$, in

some vertex of G. This is because the homotopy types of C_i and D_j on torus are distinct. The cycles C_i and D_j, however, could have many "zigzag" intersections, and besides, D_j may "wind" many times in the direction orthogonal to C_i. These problems will be dealt with in the coming proof.

First, we can assume that among all possible choices of the collection $C_1, \ldots, C_p$, we have gotten one which minimizes $|E(C_+) \setminus E(D_+)|$. An *$F$-ear* is a path having both ends in a subgraph F, but otherwise disjoint from F. Then the following is true for our choice:

Claim 4.3. No C_+-ear contained in D_+ has both ends on the same cycle C_i.

Indeed, if a C_+-ear $P \subset D_+$ with both ends on some C_i contradicted our claim, we could rectify the cycle C_i by following P in the appropriate section, thus decreasing the value of $|E(C_+) \setminus E(D_+)|$.

We further assume that the cycles $C_1, C_2, \ldots, C_p$ appear in this cyclic order around the torus; precisely, that for none $2 < i < i' \le p$ the cycles C_1 and C_i share a face in the toroidal (sub)embedding of $C_1 \cup C_2 \cup C_i \cup C_{i'}$. A *quasicycle* is a graph-homomorphic image of a cycle without degree-1 vertices, implicitly retaining its cyclic ordering of vertices. Consider an arbitrary quasicycle D'_j in G homotopic to D_1 (say, initially $D'_j = D_j$). We say that D'_j is *C_+-ear good* if (cf. Claim 4.3) no C_+-ear of D'_j has both ends on the same C_i.

With respect to the chosen quasicycle D'_j, we define an *intersection sequence* $a(j,i)$, $i = 1, \ldots, s_j$, of integers such that D'_j intersects all the C-cycles in the cyclic order $C_1 = C_{a(j,1)}, C_{a(j,2)}, \ldots, C_{a(j,s_j)}$, choosing appropriately s_j and the same orientation as with $C_1, \ldots, C_p$. We denote by $Q_{j,t}$, $t = 1, 2, \ldots, s_j$, the path of D'_j (possibly a single vertex) forming the corresponding intersection with the cycle $C_{a(j,t)}$, and by $T_{j,t}$ the path of D'_j between $Q_{j,t}$ and $Q_{j,t+1}$. Clearly, $a(j,t+1) \neq a(j,t)$ if D'_j is C_+-ear good, and hence $|a(j,t+1) - a(j,t)| \in \{1, p-1\}$ for $t = 1, 2, \ldots, s_j$.

A collection of C_+-ear good quasicycles $D'_1, D'_2, \ldots, D'_q$ in G is *quasigood* if it satisfies the property that whenever D'_n intersects D'_m in a path P (counting also the case of a self-intersection with $m = n$), the following hold up to symmetry between n and m: $P \subseteq Q_{n,x}$ for an appropriate index x of the intersection sequence of D'_n for which $a(n, x-1) = a(n, x+1)$ and $a(n,x) - a(n, x-1) \in \{1, 1-p\}$, and the adjacent paths $T_{n,x-1}$,$Q_{n,x}$,$T_{n,x}$ of D'_n stay locally on one side of the drawing of D'_m in $\mathcal{S}_1$. Informally, this means that if D'_n intersects D'_m in P, then D'_n makes a $C_{a(n,x-1)}$-ear with P "touching" D'_m from the left side. For further reference we say that D'_n is locally on the *left side* of the intersection P.

Among all choices of a quasigood collection $D'_1, D'_2, \ldots, D'_q$ in G, we select one minimizing $s_1 + \ldots + s_q$ where s_j is the above length of the intersection sequence for D'_j.

Claim 4.4. For all $1 \le j \le q$ the intersection sequence of D'_j satisfies $a(j, t-1) \neq a(j, t+1)$ for any $1 < t \le s_j$. Consequently, $D'_1, D'_2, \ldots, D'_q$ is a collection of pairwise disjoint proper cycles in G.

The idea of a proof of this claim is simple—if $a(j,t-1) = a(j,t+1)$, then we could rectify D'_j by following $C_{a(j,t-1)}$ instead of $T_{j,t-1} \cup Q_{j,t} \cup T_{j,t}$; decreasing s_j by 2. We make this formally precise now.

Let $\mathcal{R}_i$ denote the (sub)cylinder of $\mathcal{S}_1$ between the boundaries C_i and C_{i+1}. Notice that if $a(j,t-1) = a(j,t+1)$ happens for $a(j,t) - a(j,t-1) \in \{-1, p-1\}$, then neccesarily for some other index t' it holds $a(j,t'-1) = a(j,t'+1)$ and $a(j,t') - a(j,t'-1) \in \{1, 1-p\}$. So suppose for a contradiction that $a(j,t-1) = a(j,t+1) = i$ and $a(j,t) = i+1$. Then the path $P = T_{j,t-1} \cup Q_{j,t} \cup T_{j,t}$ is drawn in $\mathcal{R}_i$ with both ends on C_i and "touching" C_{i+1}. We denote by $R_0 \subset \mathcal{R}_i$ the open region bounded by P and C_i, and by P' the section of the boundary of R_0 not belonging to D'_j.

Assuming that R_0 is minimal possible over all choices of j for which $a(j,t-1) = a(j,t+1)$, we show that no D'_m, $m \in \{1,\ldots,q\}$ enters R_0: If some D'_m intersected R_0, then D'_m could not enter R_0 across P by the "stay on one side" property of a quasigood collection. Hence D'_m should enter and leave R_0 across $P' \subseteq C_i$, but not touch $Q_{j,t} \subseteq C_{i+1}$ by minimality of R_0. So D'_m would make a C_+-ear with both ends on C_i, contradicting the assumption that D'_m is C_+-ear good.

Now we form D^o_j as the symmetric difference of D'_j with the boundary of R_0 (hence D^o_j follows P'). To argue that $D'_1,\ldots,D^o_j,\ldots,D'_q$ is a quasigood collection again, it suffices to verify all possible new intersections of D^o_j along P'. So suppose there is D'_n such that its intersection $Q_{n,x}$ with C_i contains some internal vertex of P'. Since D'_n is disjoint from (open) R_0, it will "stay on one side" of D^o_j. If $Q_{n,x}$ intersects D'_j, then D'_n must be locally on the left side of this intersection, and so it is also on the left side of the intersection with new D^o_j according to the above definition. If, on the other hand, $Q_{n,x}$ is disjoint from D'_j, then the adjacent paths $T_{n,x-1}$ and $T_{n,x}$ have to connect to C_{i-1} by Claim 4.3, and so we have $a(n,x) = i$ and $a(n,x-1) = a(n,x+1) = i-1$ as required by the definition for D'_n on the left side. Hence Claim 4.4 is proved.

Claim 4.5. There is a collection of pairwise disjoint cycles $D''_1, D''_2, \ldots, D''_q$ in G where $D''_j \subset D'_j \cup C_j$, $j = 1,2,\ldots,q$, such that the cyclic intersection sequence of each D''_j is $a(j,1) = 1, a(j,2) = 2, \ldots, a(j,p) = p$ of length p.

By Claim 4.4 the intersection sequence of each D'_j has a "nice" form $a(j,1) = 1, a(j,2) = 2, \ldots, a(j,p) = p,\ a(j,p+1) = 1, \ldots$ Our task is (unless already true) to "shortcut" each D'_j such that it "winds only once" in the direction orthogonal to the loop γ. First notice that, for all $i = 1,\ldots,p$, every C_i-ear of each D'_j is C_i-separated (cf. Section 3) by Claim 4.4. We implicitly *orient* every C_i-ear so that it intersects C_{i+1} before C_{i+2}. If we take any C_1-ear $T_1 \subset D'_1$ with start x_1 and end y_1 on C_1, and any one $W_1 \subset C_1$ of the two paths between x_1, y_1, then the cycle $D''_1 = T_1 \cup W_1$ has the desired intersection sequence.

Secondly, notice that since D''_1 is not homotopic to D'_1, every D'_j has to intersect D''_1 in W_1. We may assume that the cycles $D'_2,\ldots,D'_q$ have this ordering of their first intersections with W_1 from x_1. Now for $j = 2,3,\ldots,q$ we do: let $Q_{j,x}$ be the intersection of D'_j with W_1 closest to x_1, and let $T'_j \subset D'_j$ be the unique

C_1-ear starting at $Q_{j,x}$. Then let $T_j \subset D'_j$ be the unique C_j-ear starting inside T'_j (and hence not intersecting W_1), and $W_j \subset C_j$ be the path between the ends of T_j disjoint from T_1. We set $D''_j = T_j \cup W_j$. It is straightforward to verify that $D''_1, \ldots, D''_q$ is a collection of pairwise disjoint cycles in G.

Finally, with Claim 4.5 at hand it is easy to finish the whole theorem: contractions of all the paths of $D''_j \cap C_i$, $1 \le i \le p$, $1 \le j \le q$, into single vertices, create a subdivision of the $q \times p$ toroidal grid in G. ∎

5 Conclusions

We observe that the apparent "weakness" of our approximation (Theorem 3.3) in requiring large dual edge-width of $\bar{G}$ with respect to Δ is unavoidable. Indeed, a toroidal embedded graph of dual edge-width $k = 2$ may easily be planar. By multiplying edges of such a graph and some local modification one can get (multi)graphs of crossing number one but arbitrarily large dual edge-width on the torus, at the expense of growing Δ.

It is natural to ask whether our results can be extended to higher genus surfaces. The upper bound techniques, as worked out in [3] or [6], seem to provide a road map for such an extension: Specifically, for G embedded on the orientable surface $\mathcal{S}_g$, we can iterate g-times the "cut and open" construction from Algorithm 2.3. Denoting by k_i the dual edge-width and by ℓ_i the associated orthogonal width obtained at steps $i = 1, 2, \ldots, g$, we straightforwardly conclude with a planar drawing of G of at most $O\big(g^2 \cdot \max\{k_i \ell_i : i = 1, \ldots, g\}\big)$ crossings. On the other hand, a nontrivial lower bound of order $\Omega(k_g \ell_g / \Delta^2)$ is easy to obtain using Theorem 3.3 at the last iteration. Unfortunately, this bound generally falls way short of matching the upper bound within a constant factor, even with fixed g and Δ. We have not yet been able to find a remedy for this problem.

Finally, we remark that our "grid" Theorem 3.3 itself seems to be of some interest in structural topological graph theory.

References

1. Bokal, D., Fijavz, G., Mohar, B.: The minor crossing number. SIAM Journal on Discrete Mathematics 20, 344–356 (2006)
2. Bokal, D., Fijavz, G., Wood, D.R.: The Minor Crossing Number of Graphs with an Excluded Minor. (2007), Preprint, `http://arxiv.org/math/0609707`
3. Böröczky, K., Pach, J., Tóth, G.: Planar crossing numbers of graphs embeddable in another surface. Internat. J. Found. Comput. Sci. 17, 1005–1015 (2006)
4. Brunet, R., Mohar, B., Richter, R.B.: Separating and nonseparating disjoint homotopic cycles in graph embeddings. J. of Combinatorial Theory ser. B 66, 201–231 (1996)
5. Cabello, S., Mohar, B.: Finding shortest non-separating and non-contractible cycles for topologically embedded graphs. Discrete Comput. Geom. 37, 213–235 (2007)
6. Djidjev, H., Vrt'o, I.: Planar crossing numbers of genus g graphs. In: Bugliesi, M., Preneel, B., Sassone, V., Wegener, I. (eds.) ICALP 2006. LNCS, vol. 4051, pp. 419–430. Springer, Heidelberg (2006)

7. de Graaf, M., Schrijver, A.: Grid Minors of Graphs on the Torus. J. of Combinatorial Theory ser. B 61, 57–62 (1994)
8. Even, G., Guha, S., Schieber, B.: Improved Approximations of Crossings in Graph Drawings and VLSI Layout Areas. SIAM J. Comput. 32(1), 231–252 (2002)
9. Garey, M.R., Johnson, D.S.: Crossing number is NP-complete. SIAM J. Algebraic Discrete Methods 4, 312–316 (1983)
10. Gitler, I., Hliněný, P., Leaños, J., Salazar, G.: The crossing number of a projective graph is quadratic in the face–width. In: Extended abstract in EuroComb'07, ENDM 29C, pp. 219–223 (submitted, 2007)
11. Grohe, M.: Computing Crossing Numbers in Quadratic Time. J. Comput. Syst. Sci. 68, 285–302 (2004)
12. Hliněný, P.: Crossing number is hard for cubic graphs. J. of Combinatorial Theory ser. B 96, 455–471 (2006)
13. Hliněný, P., Salazar, G.: On the Crossing Number of Almost Planar Graphs. In: Kaufmann, M., Wagner, D. (eds.) GD 2006. LNCS, vol. 4372, pp. 162–173. Springer, Heidelberg (2007)
14. Juárez, H.A., Salazar, G.: Drawings of $C_m \times C_n$ with one disjoint family II. J. of Combinatorial Theory ser. B 82, 161–165 (2001)
15. Kawarabayashi, K., Reed, B.: Computing crossing number in linear time. In: Symposium on Theory of Computing 2007, pp. 382–390. ACM Press, New York (2007)
16. Kutz, M.: Computing shortest non-trivial cycles on orientable surfaces of bounded genus in almost linear time. In: Annual Symposium on Computational Geometry 2006, pp. 430–438. ACM Press, New York (2006)
17. Mohar, B.: A linear time algorithm for embedding graphs in an arbitrary surface. SIAM J. Discrete Math. 12, 6–26 (1999)
18. Mohar, B.: On the crossing number of almost planar graphs. Informatica 30, 301–303 (2006)
19. Mohar, B., Thomassen, C.: Graphs on surfaces. Johns Hopkins Studies in the Mathematical Sciences. Johns Hopkins University Press, Baltimore MD, USA (2001)
20. Garcia–Moreno, E., Salazar, G.: Bounding the Crossing Number of a Graph in terms of the Crossing Number of a Minor with Small Maximum Degree. J. Graph Theory 36, 168–173 (2001)
21. Pach, J., Tóth, G.: Crossing numbers of toroidal graphs. In: Healy, P., Nikolov, N.S. (eds.) GD 2005. LNCS, vol. 3843, pp. 334–342. Springer, Heidelberg (2006)
22. Riskin, A.: The crossing number of a cubic plane polyhedral map plus an edge. Studia Sci. Math. Hungar. 31, 405–413 (1996)
23. Telle, J.A., Wood, D.: Planar decompositions and the crossing number of graphs with an excluded minor. In: Kaufmann, M., Wagner, D. (eds.) GD 2006. LNCS, vol. 4372, pp. 150–161. Springer, Heidelberg (2007)

Width-Optimal Visibility Representations of Plane Graphs

Jia-Hao Fan[1], Chun-Cheng Lin[1], Hsueh-I Lu[2], and Hsu-Chun Yen[1,3]

[1] Dept. of Electrical Engineering, National Taiwan University, Taipei, Taiwan, ROC
{grant,sanlin}@cobra.ee.ntu.edu.tw, yen@cc.ee.ntu.edu.tw
[2] Dept. of Computer Science and Information Engineering, National Taiwan University, Taipei, Taiwan, ROC
hil@csie.ntu.edu.tw
[3] Dept. of Computer Science, Kainan University, Taoyuan, Taiwan, ROC

Abstract. Given an n-node plane graph G, the *visibility representation* of G is concerned with drawing each node of G using a horizontal line segment such that the line segments associated with any two adjacent nodes of G are vertically visible to each other. Finding most compact visibility representations of plane graphs is not only of theoretical importance but also of practical interest, and has received much attention in the community of algorithmic graph theory. In this paper, we give a linear-time algorithm to find a visibility representation of G no wider than $\lfloor \frac{4n}{3} \rfloor - 2$. Our result improves upon the previously known upper bound $\frac{4n}{3} + 2\lceil \sqrt{n} \rceil$, providing a positive answer to a conjecture suggested in the literature about whether an upper bound $\frac{4n}{3} + O(1)$ on the required width can be achieved for an arbitrary plane graph. In fact, our visibility representation achieves optimality in the upper bound of width because the bound differs from the previously known lower bound $\lfloor \frac{4n}{3} \rfloor - 3$ only by one unit.

1 Introduction

Let G be an n-node plane graph. A *visibility representation* of G represents each node of G as a horizontal line segment, called a *node segment*, such that the node segments representing any two adjacent nodes of G are vertically visible to each other (see Figure 1). Computing *compact* visibility representations of planar graphs gets a lot of attention in the literature because it has many applications in algorithmic graph theory [1,10] as well as VLSI layout design [8]. As far as the area size of a visibility representation is concerned, one may apply the convention of locating the endpoints of node segments on the grid points of an integer grid. The *compactness* of a visibility representation is typically measured in terms of the *width* and the *size* of the smallest bounding rectangle of the visibility representation on the grid.

Otten and van Wijk [6] gave the first known algorithm for visibility representations of planar graphs, although their algorithm reveals no bound for the

T. Tokuyama (Ed.): ISAAC 2007, LNCS 4835, pp. 160–171, 2007.

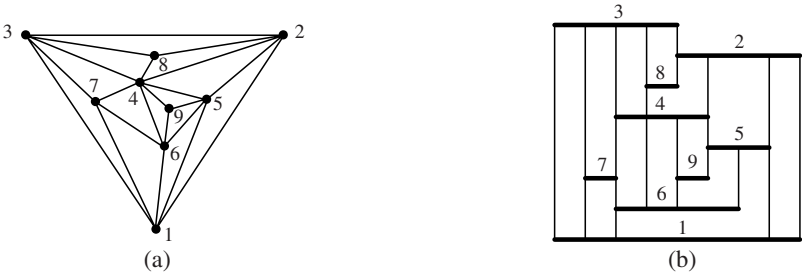


Fig. 1. (a) A plane triangulation and (b) its visibility representation

compactness of the output. Rosenstiehl and Tarjan [7], Tamassia and Tollis [9], and Nummenmaa [5] independently proposed $O(n)$-time algorithms whose outputs are no wider than $2n-5$. Lin et al. [4] improved the required width to $\lfloor \frac{22n-42}{15} \rfloor$, and conjectured that any n-node plane graph G has a visibility representation no wider than $\frac{4n}{3} + O(1)$. Zhang and He [11] further improved the required width to $\lfloor \frac{13n-24}{9} \rfloor$. As for the required height, a visibility representation of G with height no more than $n-1$ was reported in [7,9]. Zhang and He [10,12,13] reduced the required height to $\lfloor \frac{15n}{16} \rfloor$, then $\lfloor \frac{5n}{6} \rfloor$, and finally $\lfloor \frac{4n-1}{5} \rfloor$. Very recently, He and Zhang [2] proved that every plane graph has a visibility representation with height at most $\frac{2n}{3} + 2\lceil \sqrt{n/2} \rceil$ and a visibility representation with width at most $\frac{4n}{3} + 2\lceil \sqrt{n} \rceil$.

As for the required size, Zhang and He [12] showed that any visibility representation of a planar graph requires a size at least $\lfloor \frac{2n}{3} \rfloor \times (\lfloor \frac{4n}{3} \rfloor - 3)$, which provides a positive answer to Kant's open question [3] about whether there exists a plane graph such that all of its visibility representations require width greater than $c \times n$, where $c > 1$.

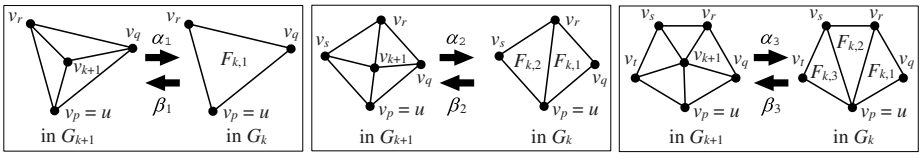
In this paper, we devise an $O(n)$-time algorithm to obtain a visibility representation of a given plane graph with its width no wider than $\lfloor \frac{4n}{3} \rfloor - 2$. Since the bound differs from the lower bound $\lfloor \frac{4n}{3} \rfloor - 3$ (reported in [12]) only by a unit, the visibility representations produced by our algorithm achieve optimality in terms of the width of the drawing area. In addition, our result improves upon the previously known upper bound $\frac{4n}{3} + 2\lceil \sqrt{n} \rceil$, answering in the affirmative a conjecture (by Lin et al. [4]) about whether an upper bound $\frac{4n}{3} + O(1)$ on the required width can be achieved for an arbitrary plane graph.

2 Constructive Ordering of a Plane Triangulation

Without loss of generality, we assume that the input graph $G = (V, E)$ is an n-node *plane triangulation* for $n \geq 3$. A *plane graph* is a planar graph associated with a fixed planar embedding. The embedding of a plane graph divides the plane into a number of connected regions, each of which is called a *face*. The unbounded face of G is called the *outer face*, whereas the remaining faces are *inner faces*. G is a *plane triangulation* if G has at least three nodes and

the boundary of each face, including the outer face, is a triangle. In a *visibility drawing* (a.k.a., *visibility representation*) of G, every node of G is represented by a horizontal line segment, called a *node segment*, such that the node segments representing any two adjacent nodes of G are connected by a vertical line segment, called an *edge segment*. The *visibility embedding* of a drawing of G associates to each node v a circular counterclockwise ordering of the incidence list (set of neighbors) $Adj(v)$ of v which is divided into two sublists: one is for the neighbor nodes which are visible to v from the down side (denoted by $Adj_1(v)$); the other sublist includes the other nodes (denoted $Adj_2(v)$). For example, in Figure 1(b), Adj_1(node 4) = nodes $2, 8, 3$ and Adj_2(node 4) = nodes $7, 6, 9, 5$. Throughout the paper, the degree of a node v is denoted $deg(v)$; the x-coordinate of the left (resp. right) endpoint of the node segment representing v in a visibility drawing is denoted $x^-(v)$ (resp. $x^+(v)$); and the y-coordinate of the node segment representing v in the visibility drawing is denoted $y(v)$.

For ease of explanation, we define the so-called *coalescing* and *splitting* operations of v_{k+1} with degree three, four, and five as follows. It suffices to define the operations for degree five; the other operations are similar and simpler. In Figure 2 (c), we say that the case when G_{k+1} becomes G_k is due to *coalescing* two nodes v_{k+1} and u, whereas the case when G_k becomes G_{k+1} is due to *splitting* node v_{k+1} from the common node u of faces $F_{k,1}$, $F_{k,2}$, and $F_{k,3}$. Note that a splitting operation also can be regarded as either *attaching* two new faces to node u or *inserting* a node at faces $F_{k,1}$, $F_{k,2}$, and $F_{k,3}$. We denote by $\alpha_3(v_{k+1}, u)$ the operation of coalescing v_{k+1} and u, whereas the operation of splitting v_{k+1} from u at faces $F_{k,1}, F_{k,2}, F_{k,3}$ is denoted by $\beta_3(v_{k+1}, F_{k,1}, F_{k,2}, F_{k,3})$. In a similar way, $\alpha_1(v_{k+1}, u)$, $\beta_1(v_{k+1}, F_{k,1})$ for $deg(v_{k+1}) = 3$ as well as $\alpha_2(v_{k+1}, u)$, $\beta_2(v_{k+1}, F_{k,1}, F_{k,2})$ for $deg(v_{k+1}) = 4$ are defined.



(a) $deg(v_{k+1}) = 3$ in G_{k+1} (b) $deg(v_{k+1}) = 4$ in G_{k+1} (c) $deg(v_{k+1}) = 5$ in G_{k+1}

Fig. 2. Illustration of the coalescing and splitting operations

Let $\pi = (v_1, v_2, ..., v_n)$ be an ordering of all nodes in G, in which v_1, v_2, v_3 are the three nodes of the outer face of G counterclockwise. For each integer k, $3 \leq k \leq n-1$, the plane graph G_{k+1} involving the $k+1$ nodes $v_1, v_2, ..., v_k, v_{k+1}$ is produced by splitting v_{k+1} from some node in G_k. Let $G_n = G$. We call π a *constructive ordering* of a plane triangulation G if the following conditions hold for each k, $3 \leq k \leq n$:

(c1) G_k is a plane triangulation with outer edges v_1v_2, v_2v_3, v_3v_1;

(c2) if $3 \leq k \leq n-1$, then node v_{k+1} is split from a node in G_k, and the degree of node v_{k+1} is three, four, or five in G_{k+1}.

We have the following lemma.

Lemma 1. *Every n-node plane triangulation G has a constructive ordering, which can be found in $O(n)$ time.*

3 Our Width-Optimal Drawing Algorithm

The section shows the following main result of this paper.

Theorem 1. *Given an n-node plane triangulation G, a visibility representation of G with its width bounded by $\lfloor \frac{4n}{3} \rfloor - 2$ can be obtained in time $O(n)$.*

Consider a constructive ordering $\pi = (v_1, v_2, ..., v_n)$ of G. Recall that, for $k = 3$ to $n-1$, G_{k+1} involving $k+1$ nodes $v_1, v_2, ..., v_k, v_{k+1}$ is produced by splitting v_{k+1} from some node in G_k. In our drawing algorithm, for $k \in \{3, 4, ..., n\}$, each inner face in G_k is drawn as an L-shape depicted in Figure 3, which involves four cases depending on whether the bottom node segment is as wide as the top node segment and where the middle node segment is placed. The one with the bottom node segment wider than the top node segment is called a *regular L-shape*; otherwise, *degenerated L-shape*. The one with the middle node segment placed on the left side is called *left L-shape*; otherwise, *right L-shape*. As far as the width of the drawing of an inner face is concerned, the *width* of an L-shape is measured by that of its bottom node segment, i.e., the middle node segment of a degenerated L-shape is of zero width. Note that this is different from the definition of the width of a visibility drawing of G, which is measured by that of the bounding box of the drawing. If we only consider the visibility embedding of an L-shape, then the regular and the degenerated L-shapes with the same relationship of y-coordinates of the three node segments are viewed as the same L-shape. By doing so, any visibility drawing $D(G_k)$ of G_k in our drawing algorithm is composed of a number of L-shapes. Since G_k is always a plane triangulation, the contour of $D(G_k)$ is also an L-shape.

Fig. 3. L-shapes

Our algorithm for generating a visibility representation of G no wider than $\lfloor \frac{4n}{3} \rfloor - 2$ is stated as follows:

Algorithm Visibility
Input: a plane triangulation G
Output: a width-optimal visibility drawing of G

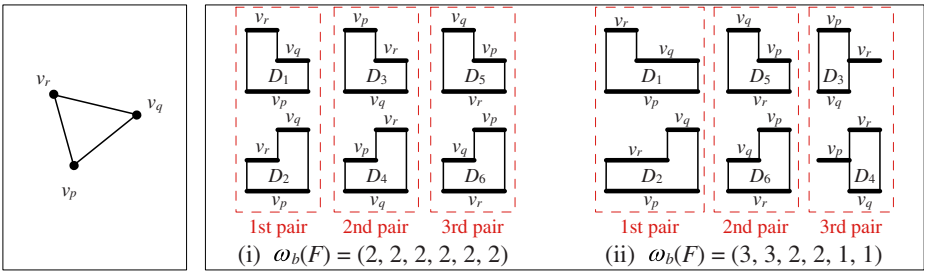
1. A constructive ordering $\pi = (v_1, v_2, ..., v_n)$ of G is obtained by Lemma 1.
2. Initially, we draw all the (six) possible (L-shaped) visibility drawings of G_3 as shown in Figure 4 (b)(i), where $(v_p, v_q, v_r) = (v_1, v_2, v_3)$. Note that the six drawings can be classified into three pairs, where each of the three nodes v_1-v_3 serves as a bottom node segment in turn in each pair. Also note that every drawing has the narrowest width (i.e., two).
3. For each node v_{k+1}, $k = 3, 4, ..., n-1$, we maintain six drawings of G_{k+1} by doing the following:
   (a) If $Deg(v_{k+1}) = 3$, then we execute the β_1 operation shown in Figure 5.
   (b) If $Deg(v_{k+1}) = 4$, then we execute the β_2 operation shown in Figure 6, depending on the shapes of drawings of $F_{k,1}$ and $F_{k,2}$ in each of six drawings of G_k.
   (c) If $Deg(v_{k+1}) = 5$, then we execute the β_3 operation shown in Figure 7, depending on the shapes of drawings of $F_{k,1}$, $F_{k,2}$, and $F_{k,3}$ in each of six drawings of G_k.

   Appropriately adjust the drawings of the other faces on the left, right, top, and bottom of the drawings transformed from the drawings of $F_{k,1}$, $F_{k,2}$, and $F_{k,3}$.
4. Compress the width of each of the six drawings of $G_n = G$ as much as possible. Then output the drawing of G with the narrowest width.

Note that each of Figures 6 (a) and (b) includes four possible cases depending on that the two input L-shapes are left (denoted l) or right (denoted r) L-shape, e.g., Figure 6(a)(1-ii) is the case when the configuration of the two input L-shapes is rl, which means that from the left to the right the first input L-shape is a right L-shape and the second input L-shape is a left L-shape. According to this classification, we demonstrate all possible cases of the β_2 operation in Figure 6. In a similar way, all possible cases of the β_3 operation are demonstrated in Figure 7. As a result, we have the following observation, upon which Step 3 of Algorithm **Visibility** is based.

Observation 1. *If every inner face in G_k is drawn as an L-shape, then Figure 5 (resp., Figure 6 and Figure 7) illustrates all the possible cases of the β_1 (resp., β_2 and β_3) operation. Furthermore, the bottom node segment of each of the six drawings of any face rather than $F_{k,1}$–$F_{k,3}$ is not modified in executing the operation.*

Before showing the proof of the main theorem in detail, some notations are given as follows. Recall that, in Figure 3, the width of L-shape $D(F)$ of an inner face F is determined by that of its bottom node segment, so we denote by $w_b(D(F))$ the width of the bottom node segment of $D(F)$. Note that each



(a) A face F in G_k (b) Six drawings of a good face with the narrowest width

Fig. 4. Six possible visibility drawings for a face $v_p v_q v_r$

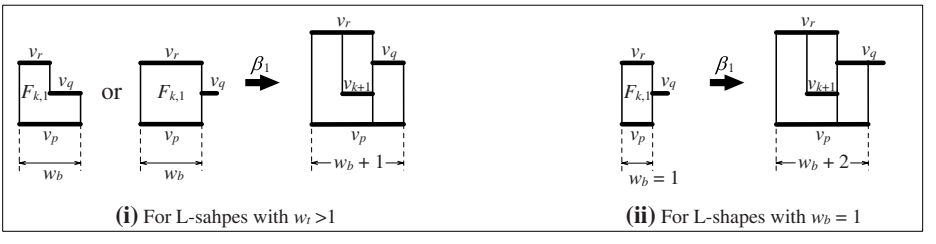
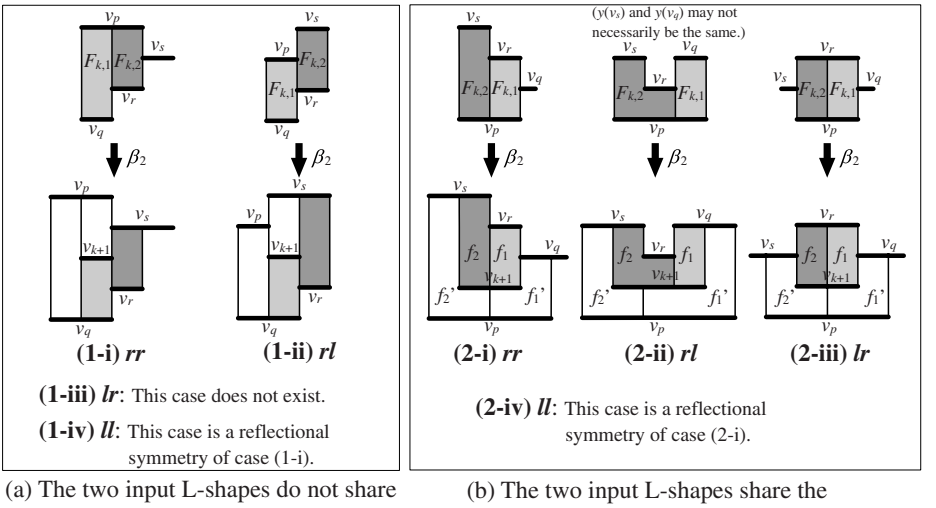


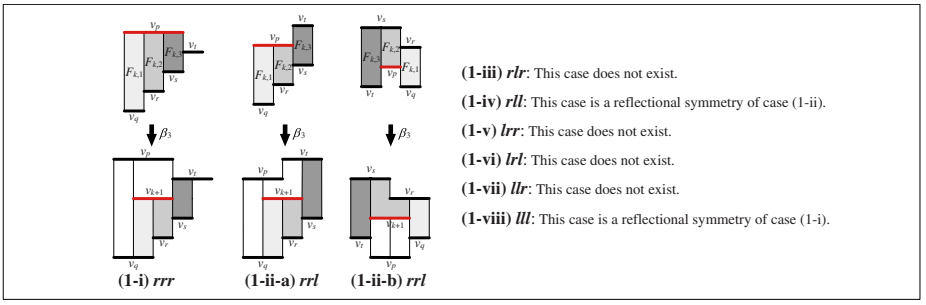
Fig. 5. The splitting operation of node v_{k+1} with degree three



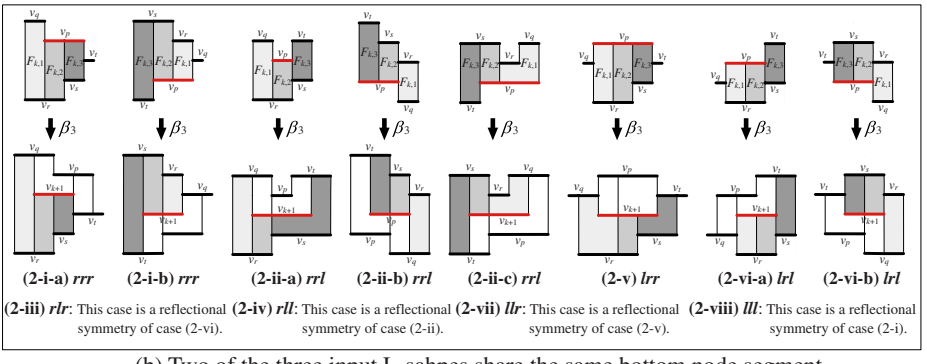
(a) The two input L-shapes do not share the same bottom node segment.

(b) The two input L-shapes share the same bottom node segment.

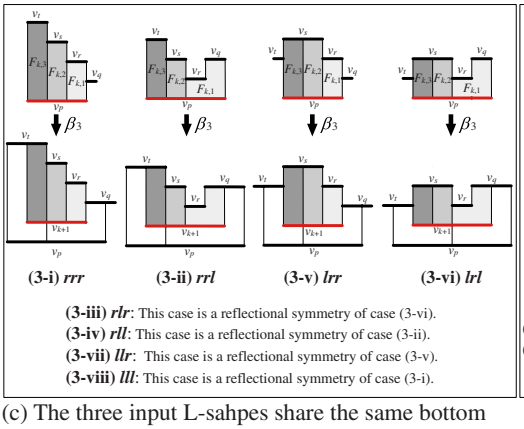
Fig. 6. The splitting operation of node v_{k+1} with degree four



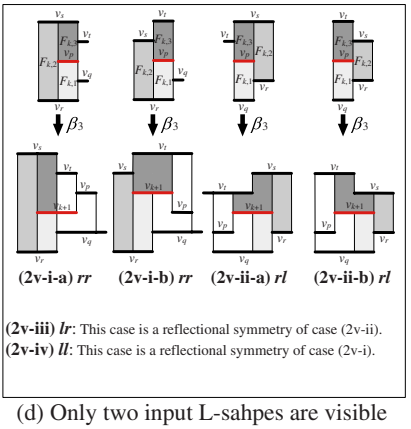
(a) None of the three input L-shapes shares the same bottom node segment.



(b) Two of the three input L-sahpes share the same bottom node segment.



(c) The three input L-sahpes share the same bottom node segment.



(d) Only two input L-sahpes are visible from the down side.

Fig. 7. The splitting operation of node v_{k+1} with degree five

iteration of Step 3 in our algorithm maintains six visibility drawings of G_{k+1}. Let L denote a face or a subgraph in G_{k+1}. The L-shapes of L appearing in the six drawings are denoted by $D_1(L)$-$D_6(L)$, and $\mathfrak{D}(L) = \{D_1(L), D_2(L), ..., D_6(L)\}$.

Then we define $\omega_b(L) = (w_b(D_1(L)), w_b(D_2(L)), ..., w_b(D_6(L)))$, and the sum of widths of the six drawings is denoted as $s(L) = \sum_{i=1}^{6} w_b(D_i(L))$. With the above notations, we have the following observation.

Observation 2. *The six drawings of every face F in $\mathfrak{D}(G_{k+1})$ produced by Step 3 of Algorithm* **Visibility** *can be classified into three pairs, where each of the three nodes of F serves as a bottom node segment in one of the three pairs.*

By the above observation, the six visibility drawings $D_1(G_{k+1})$-$D_6(G_{k+1})$ of G_{k+1} can be classified into three pairs, in which we say that $D_{2i-1}(G_{k+1})$ and $D_{2i}(G_{k+1})$ are the i-th *pair* of drawings, $1 \leq i \leq 3$, and each of the three nodes of any face in G_{k+1} serves as a bottom node segment in turn in each pair. If node v serves as a bottom node segment in a certain pair, then we say that the pair is *based* on node v. Note that the ordering of pairs does not matter. For example, the first pair of a certain face in $\mathfrak{D}(D_k)$ may be said to be the second pair of the face in $\mathfrak{D}(G_{k+1})$. In fact, since our algorithm starts from the drawings shown in Figure 4(b)(i), the two drawings of the same pair in $\mathfrak{D}(G_{k+1})$ produced by Step 3 of our algorithm are almost the same in visualization except for the two topmost node segments, and they have the same width. As a consequence, we only need to consider $D_1(L)$, $D_3(L)$, and $D_5(L)$ in the following discussion.

Not only the six drawings of any face can be classified into three pairs, the input L-shapes of any of the β_1, β_2, and β_3 operations also can be classified into three pairs according to Observation 2 and the nature of the input L-shapes. The classification is summarized in Table 1, in which we assume that the nodes appearing counterclockwise on the contour of the input adjacent faces $F_{k,1}$–$F_{k,3}$ (resp., $F_{k,1}$–$F_{k,2}$; $F_{k,1}$) for the β_3 (resp., β_2; β_1) operation in G_k are denoted by u_1–u_5 (resp., u_1–u_4; u_1–u_3), where u_1 is the common node of faces $F_{k,1}$–$F_{k,3}$ ($F_{k,1}$–$F_{k,2}$; none). Note that there are three possible cases for the β_3 operations in Table 1, and one can check that the union of the three cases just corresponds to all the possible cases of our β_3 operation.

Two adjacent L-shapes are said to form a *U-shape* if they share the same middle and bottom node segments (e.g., see the top drawing in Figure 6(2-ii)). Given i adjacent L-shapes sharing the same bottom node segment in a certain pair of visibility drawings, a degree-$(i+2)$ *U-shaped insertion* is to attach two new L-shapes forming a U-shape to the bottom node segment, as shown in Figure 8(b). For convenience, the degree-$(i+2)$ U-shaped insertion is also called the μ_i operation. As executing a μ_i operation in a pair of drawings (Figure 8(b)), the operation in the other two pairs must be of the case shown in Figure 8(c) by Observation 2. Comparing Table 1 with Figure 8, one can observe that the β_1 operation, the β_2 operation, and the first case of the β_3 operation in Table 1 are just of the U-shaped insertions. Since we observe from Figure 8 that in general the width vector ω_b of the six drawings of G_k is increased by $(+2, +2, +1, +1, +1, +1)$ after executing a U-shaped insertion, hence, $s(G_{k+1}) - s(G_k) = 8$. As a result, if our six visibility drawings are produced only by U-shaped insertions, then the drawing with the narrowest width among the six ones must be no wider than the average of the total sum of widths, i.e., $\lfloor \frac{12+8\times(n-3)}{6} \rfloor = \lfloor \frac{4n}{3} \rfloor - 2$ (since $s(G_3) = 12$ and there are $(n-3)$ insertions), as required.

Table 1. The three pairs of the input L-shapes (the six drawings) of the β_1, β_2, and β_3 operations

	1st pair	2nd pair	3rd pair
β_1 or μ_1	Figure 5: $v_p = u_1$	Figure 5: $v_p = u_2$	Figure 5: $v_p = u_3$
β_2 or μ_2	Figure 6(b): $v_p = u_1$	Figure 6(a): $v_r = u_3$, $v_q = u_4$	Figure 6(a): $v_r = u_3$, $v_q = u_2$
β_3 or μ_3	Figure 7(c): $v_p = u_1$	Figure 7(a) excluding (1-ii-b): $(v_q, v_r, v_s) = (u_2, u_3, u_4)$	Figure 7(a) excluding (1-ii-b): $(v_q, v_r, v_s) = (u_5, u_4, u_3)$
β_3	Figure 7(b) (2-i-b), (2-ii-c): $(v_p, v_t) = (u_1, u_5)$; Figure 7(b) (2-ii-b), (2-vi-c): $(v_p, v_q) = (u_1, u_5)$	Figure 7(d) (2v-ii-a), (2v-ii-b): $(v_p, v_q, v_r) = (u_1, u_2, u_3)$	Figure 7(b) (2-i-a), (2-ii-a), (2-v), (2-vi-a): $\{v_r, v_s\} = \{u_3, u_4\}$
β_3	Figure 7(d) (2v-i-a), (2v-i-b): $(v_p, v_r) = (u_1, u_4)$	Figure 7(d) (2v-i-a), (2v-i-b): $(v_p, v_r) = (u_1, u_3)$	Figure 7(a)(1-ii-b): $v_p = u_1, \{v_q, v_t\} = \{u_2, u_5\}$

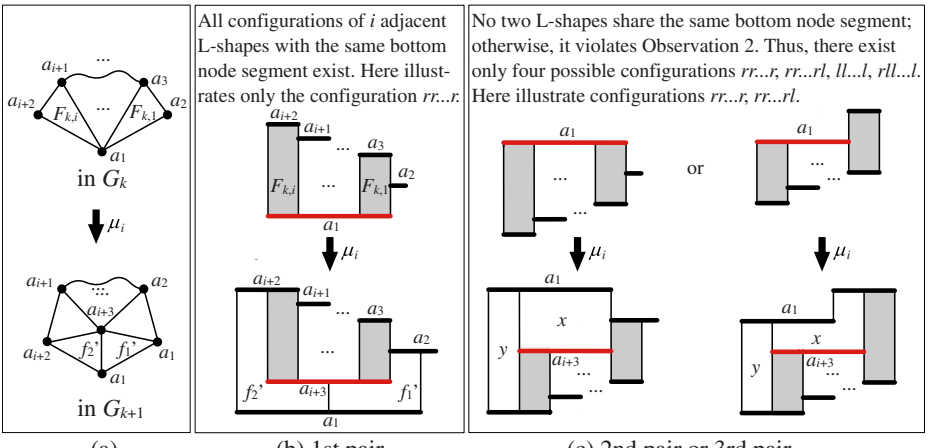


Fig. 8. Illustration of the μ_i operation

With respect to $\mathfrak{D}(G)$, a constructive ordering is called *U-shaped constructive ordering* if a drawing in $\mathfrak{D}(G)$ is constructed only by U-shaped insertions. Fortunately, we have the following observation.

Observation 3. *There exists a U-shaped constructive ordering with respect to* $\mathfrak{D}(G)$, *produced by Step 3 of Algorithm* **Visibility**.

From Figure 5(ii), the drawing width is increased by two units after executing a μ_1 operation (β_1 operation) at an L-shape with $w_b = 1$. Therefore, if the μ_1 operation is executed at a face F where the number of pairs of the L-shapes with $w_b = 1$ is at least two, then $s(G_{k+1}) - s(G_k) \geq 10 > 8$, which is not our required result. As a result, in order to guarantee that $s(G_{k+1}) - s(G_k) = 8$, we require to adjust the edge segments such that every μ_1 operation is always

executed at a so-called *good* face. We say that a face F is *good* in $\mathfrak{D}(G_k)$ if $\omega_b(F) = (p, p, q, q, r, r)$ such that

- at most one of p, q, r is one;
- $p + q + r \geq 6$.

Figures 4 (b)(i) ($\omega_b = (2, 2, 2, 2, 2, 2)$) and (b)(ii) ($\omega_b = (3, 3, 2, 2, 1, 1)$) illustrate two possible cases of a good face with narrowest width. Note that we need condition $p+q+r \geq 6$ so that any face in the graph transformed after executing more than one μ_1 operation at face F can be adjusted to be good.

In our discussion, if there is a μ_1 operation executed at a face F which is not good, then face F may be adjusted to be good by *borrowing* a unit of width from the other faces (in fact, adjusting the edge segments), without changing the width of the whole drawing. After executing the μ_1 operation, the unit of width will be *returned* to its creditors. Take Figure 9 for example. Consider a μ_1 operation executed at face F_2 with $\omega_b(F_2) = (3, 3, 1, 1, 1, 1)$ (which is not a good face) in Figure 9(a). Face F_2 may *borrow* a unit of width from region $v_2v_3v_4$ (each of faces $v_5v_2v_3$ and $v_5v_3v_4$ lends a unit of width) in the second pair so that it turns out that $\omega_b(F_2) = (3, 3, 2, 2, 1, 1)$, which means that face F_2 is good. After inserting node v_6, we observe that $\omega(F_3) = (2, 2, 2, 2, 3, 3)$ in Figure 9(b), which implies that F_3 is good even if $D_5(F_3)$ is decreased by one unit of width. Therefore, face F_3 can return a unit of width to region $v_2v_3v_4$ (each of faces $v_5v_2v_3$ and $v_5v_3v_4$ obtains a unit of width), as shown in Figure 9(b). Note that a region may lent a unit of width to face F and obtain a unit of width from the other faces rather than F in different pairs. In light of the above, we realize that it is a challenging endeavor to guarantee that a sequence of μ_1 operations is always executed at good faces.

For discussing whether a sequence of μ_1 operations is always executed at good faces, we say that a *good graph* g_k, associated with six visibility drawings $\mathfrak{D}(g)$, is a graph that satisfies the following recursive conditions:

- any face in g can be adjusted to be good without changing the width of every drawing in $\mathfrak{D}(g)$; furthermore, if a face is adjusted to be good by borrowing a unit of width from a certain region, then a unit of width will be returned to the region after executing a next μ_1 operation.
- if g' is a graph transformed by executing a μ_1 operation at a good face in g, then g' is a good graph.

Note that when the μ_1 operation is executed at a good face, $s(g') = s(g) + 8$ always holds.

In order to achieve the goodness of a face in a region consisting of faces, the region may need to borrow a unit of width from the other certain region, and vice versa. In this case, we say that the two regions are *dependent*; otherwise, *independent*. We have the following lemmas (their proofs are omitted due to page limitation).

Lemma 2. *An independent good face is a good graph.*

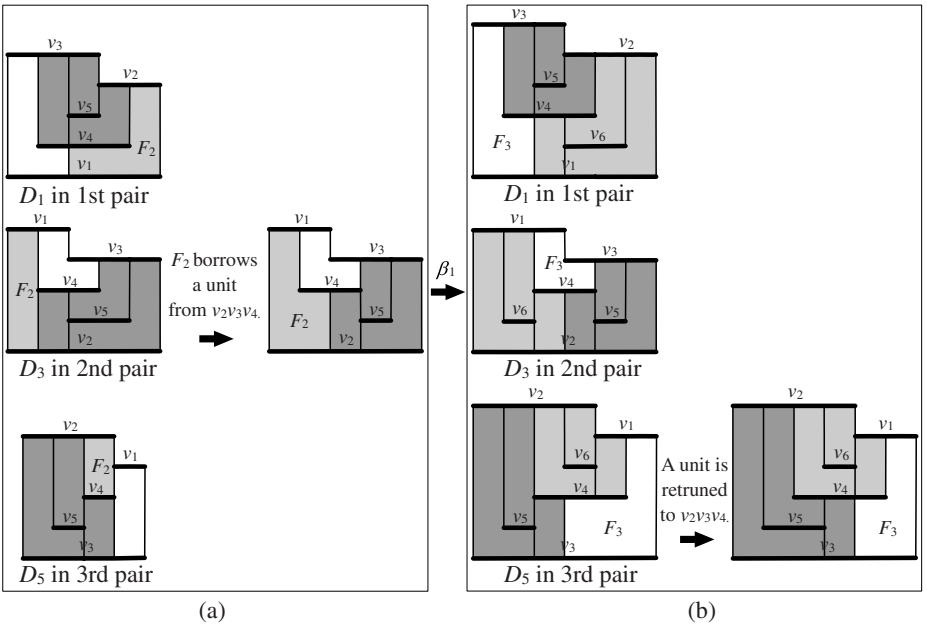


Fig. 9. Illustration of borrowing and lending a unit of width

Lemma 3. *Given a good graph g (associated with six visibility drawings $\mathfrak{D}(g)$), a graph g' transformed by executing a U-shaped insertion at g is a good graph. In addition, $s(g') = s(g) + 8$.*

Based on the above, we are ready to present the proof of Theorem 1 as follows:

(**Proof of Theorem 1**) We claim that there exists a visibility drawing D_{opt} with width $\lfloor \frac{4n}{3} \rfloor - 2$ which has the same visibility embedding Δ as the visibility drawing D_{our} produced by Algorithm **Visibility**. Since D_{our} is the visibility drawing with the minimum width under the visibility embedding Δ by Step 4 of Algorithm **Visibility**, D_{our} must be no wider than $\lfloor \frac{4n}{3} \rfloor - 2$.

Now we show the claim. By Observation 3, we have a U-shaped constructive ordering with respect to $\mathfrak{D}(G)$, produced by Step 3 of Algorithm **Visibility**. With respect to the U-shaped constructive ordering $\pi = (v_1, v_2, ..., v_n)$ for $\mathfrak{D}(G)$, we proceed by induction on $k = 3, 4, ..., n$ to show that G_k is a good graph; $s(G_k) = s(G_{k-1}) + 8$ for $k \neq 3$. Initially, the six drawings of G_3 is produced by Step 2 of Algorithm **Visibility**, as shown in Figure 4(b)(i), i.e., G_3 is a good face. By Lemma 2, G_3 is a good graph. Suppose that G_k is a good graph. By Lemma 3, G_{k+1} is a good graph and $s(G_k) = s(G_{k-1}) + 8$.

Since G_3 is a good face whose drawings are shown in Figure 4(b)(i), $s(G_3) = 6 \times 2 = 12$. Since there are $(n-3)$ U-shaped insertions, each of which contributes 8 units of width, hence, $s(G_n) = 12 + (n-3) \times 8 = 8n - 12$. As a result, the

drawing with the minimum width among the six drawings of G_n must be no wider than the average of $s(G_n)$, i.e., $\lfloor \frac{8n-12}{6} \rfloor = \lfloor \frac{4n}{3} \rfloor - 2$, as claimed.

As for the running time, it is easy to see that Step 1 and Step 2 in Algorithm **Visibility** take time $O(n)$. Step 3 can be implemented in $O(n)$ time as follows. We do not adjust the drawings of the other faces in each iteration of Step 3. We only record the relative positions of node and edge segments instead. After finishing all iterations of Step 3, we produce the six final drawings according to the information of the relative positions of node and edge segments. Step 4 can be implemented in $O(n)$ time [4]. Consequently, the total running time of Algorithm **Visibility** is $O(n)$. □

References

1. Di Battista, G., Tamassia, R., Tollis, I.G.: Constrained visibility representations of graphs. Inf. Process. Lett. 41(1), 1–7 (1992)
2. He, X., Zhang, H.: Nearly optimal visibility representations of plane graphs. In: Bugliesi, M., Preneel, B., Sassone, V., Wegener, I. (eds.) ICALP 2006. LNCS, vol. 4051, pp. 407–418. Springer, Heidelberg (2006)
3. Kant, G.: A more compact visibility representation. Internat. J. Comput. Geom. Appl. 7, 197–210 (1997)
4. Lin, C.-C., Lu, H.-I., Sun, I.-F.: Improved compact visibility representation of planar graph via Schnyder's realizer. SIAM J. Discrete Math. 18(3), 19–29 (2004)
5. Nummenmaa, J.: Constructing compact rectilinear planar layouts using canonical representation of planar graphs. Theoret. Comput. Sci. 99, 213–230 (1992)
6. Otten, R., van Wijk, J.: Graph representations in interactive layout design. In: IEEE Internat. Sympos. on Circuits and Systems, pp. 914–918 (1978)
7. Rosenstiehl, P., Tarjan, R.E.: Rectilinear planar layouts and bipolar orientations of planar graphs. Discrete Comput. Geom. 1(4), 343–353 (1986)
8. Schlag, M., Luccio, F., Maestrini, P., Lee, D.T., Wong, C.K.: A visibility problem in VLSI layout compaction. In: Preparata, F.P. (ed.) Advances in Computing Research, vol. II, pp. 259–282. JAI Press, Greenwich, CT (1985)
9. Tamassia, R., Tollis, I.G.: A unified approach to visibility representations of planar graphs. Discrete Comput. Geom. 1, 321–341 (1986)
10. Zhang, H., He, X.: Compact visibility representation and straight-line grid embedding of plane graphs. In: Dehne, F., Sack, J.-R., Smid, M. (eds.) WADS 2003. LNCS, vol. 2748, pp. 493–504. Springer, Heidelberg (2003)
11. Zhang, H., He, X.: Improved visibility representation of plane graphs. Comput. Geom. 30(1), 29–39 (2005)
12. Zhang, H., He, X.: Visibility representation of plane graphs via canonical ordering tree. Inf. Process. Lett. 96, 41–48 (2005)
13. Zhang, H., He, X.: An application of well-orderly trees in graph drawing. Int. J. Found. Comput. Sci. 17(5), 1129–1142 (2006)

Computing Upward Topological Book Embeddings of Upward Planar Digraphs*

F. Giordano[1], G. Liotta[1], T. Mchedlidze[2], and A. Symvonis[2]

[1] Università degli Studi di Perugia, Italy
{giordano,liotta}@diei.unipg.it
[2] National Technical University of Athens, Greece
{mchet,symvonis}@math.ntua.gr

Abstract. This paper studies the problem of computing an upward topological book embedding of an upward planar digraph G, i.e. a topological book embedding of G where all edges are monotonically increasing in the upward direction. Besides having its own inherent interest in the theory of upward book embeddability, the question has applications to well studied research topics of computational geometry and of graph drawing. The main results of the paper are as follows.

- Every upward planar digraph G with n vertices admits an upward topological book embedding such that every edge of G crosses the spine of the book at most once.
- Every upward planar digraph G with n vertices admits a point-set embedding on any set of n distinct points in the plane such that the drawing is upward and every edge of G has at most two bends.
- Every pair of upward planar digraphs sharing the same set of n vertices admits an upward simultaneous embedding with at most two bends per edge.

1 Introduction

A *book* consists of a line called *spine* and of k half-planes, called *pages*, having the spine as a boundary. A *book embedding* of a planar graph G is a drawing of G on a book such that the vertices are aligned along the spine, each edge is drawn in a page and shares with the spine only its end-vertices, and no two edges cross. A well-known result is that all planar graphs have a book embedding on four pages and that there exist some planar graphs requiring exactly four pages to be book embedded [28]. Thus, book embeddings of planar graphs are in general three-dimensional representations and if one wants to have a two dimensional drawing of a planar graph where all vertices are collinear, edges must be allowed to cross the spine. Drawings where spine crossings are allowed are known in the literature as *topological book embeddings* [13]. In [10] it is proved that every planar graph admits a topological book embedding in the plane such that every edge crosses the spine at most once.

* This work is partially supported by the MIUR Project "MAINSTREAM: Algorithms for massive information structures and data streams".

T. Tokuyama (Ed.): ISAAC 2007, LNCS 4835, pp. 172–183, 2007.

Motivated by parallel process scheduling problems, *upward book embeddings* of acyclic digraphs and of posets have also been widely investigated (see e.g., [1,19,20,21,26]). An upward book embedding of an acyclic digraph G is a book embedding of G such that the ordering of the vertices along the spine is a topological ordering of G. Informally, an upward book embedding is a book embedding in which the spine is vertical and the directed edges are drawn as curves monotonically increasing in the upward direction. In contrast to the result in [28] concerning the book embeddability of undirected planar graphs, the minimum number of pages required by an upward book embedding of a planar acyclic digraph is unbounded [19], while the minimum number of pages required by an upward book embedding of an upward planar digraph is not known [1,19,26]. Only some classes of upward planar digraphs requiring a constant number of pages have been established to date (see, e.g. [1,9,21]).

This paper studies the problem of computing an *upward topological book embedding* of an upward planar digraph G, i.e. a topological book embedding of G in 2 pages, where all edges are monotonically increasing in the upward direction. Besides having its own inherent interest in the theory of upward book embeddability, the question has applications to well studied research topics of graph drawing and of computational geometry. The first and more immediate application is in the context of computing drawings of hierarchical structures where it is required to consider not only aesthetic constraints such as the upwardness and the planarity but also semantic constraints expressed in terms of collinearity for a (sub)set of the vertices; for example, in the application domains of knowledge engineering and of project management, PERT diagrams are typically drawn by requiring that critical sequences of tasks be represented as collinear vertices (see, e.g., [8,27]).

Upward topological book embeddings turn out to be also a useful tool to address a classical problem of computational geometry. Let G be a planar graph with n vertices and let S be a set of n distinct points in the plane. A *point-set embedding* of G on S is a planar drawing of G where every vertex of G is mapped to a point of S. The problem of computing point-set embeddings of planar graphs such that the number of bends along the edges be a small constant is the subject of a rich body of literature (including, e.g., [3,4,18,22]). We shall discuss how to use upward topological book embeddings in order to find new results in the context of point-set embeddings of planar acyclic digraphs with the additional constraint that all edges are oriented upward.

Finally, an emerging research direction in graph drawing studies the problem of representing and visually comparing multiple related graphs which typically come from different application domains including software engineering, telecommunications, and computational biology. *Simultaneous embeddings* (see, e.g., [5,6,11,14,16]) aid in visualizing multiple relationships between the same set of objects by keeping common vertices of these graphs in the same positions. An additional contribution of this paper is to apply upward topological book embeddings in the context of simultaneous embeddings of upward planar digraphs.

More precisely, the main results in this paper can be listed as follows.

- It is proved that every upward planar digraph G with n vertices admits an upward topological book embedding such that every edge of G crosses the spine of the book at most once. We recall that it is not known how many pages may be required if the edges must be drawn upward but are not allowed to cross the spine [1,19,26]. Our result can be regarded as the upward counterpart of [10], where topological book embeddings of non-oriented planar graphs are studied.
- It is shown that every upward planar digraph G with n vertices admits a point-set embedding on any set of n distinct points in the plane, such that the drawing is upward and every edge of G has at most two bends. Similar results were previously known only for restricted families of upward planar digraphs [9].
- Let G_1 and G_2 be any two upward planar digraphs defined on the same set of n vertices. An *upward simultaneous embedding* of G_1 and G_2 is a pair of upward planar drawings $< \Gamma_1, \Gamma_2 >$ such that Γ_1 is an upward planar drawing of G_1, Γ_2 is an upward planar drawing of G_2, and for each vertex v the point representing v is the same in Γ_1 and in Γ_2.
  It is shown that every pair G_1, G_2 admits an upward simultaneous embedding $< \Gamma_1, \Gamma_2 >$ such that every edge has at most two bends. Non-directed counterparts of this result are in [11,14].

The proofs of the above results are constructive and give rise to polynomial time algorithms. In particular, the drawing algorithm to compute upward topological book embeddings is based on an incremental technique that adds a face at a time by exploiting the interplay between an *st*-numbering of the upward planar digraph given as input and an *st*-numbering of its dual digraph.

The remainder of the paper is organized as follows. Basic definitions are given in Section 2. The problem of computing upward topological book embeddings of upward planar digraphs is studied in Section 3. Upward point-set embeddings and upward simultaneous embeddings are the subject of Sections 4 and 5, respectively. Finally, conclusions and possible directions for future research can be found in Section 6. For reasons of space, proofs have been omitted and can be found in [15].

2 Preliminaries

We assume familiarity with basic graph drawing terminology [2,23,25] and recall in the following only those definitions and properties that will be extensively used in the remainder of the paper.

Let G be a digraph and let u, v be any two vertices of G; (u, v) denotes the directed edge from u to v. An *st*-digraph is a biconnected acyclic digraph with exactly one source s and exactly one sink t, and such that (s, t) is an edge of the digraph. A *planar st-digraph* is an *st*-digraph that is planar and embedded with vertices s and t on the boundary of the external face. The digraph depicted in Figure 1(a) is an example of a planar *st*-digraph.

Property 1. Let v be a vertex of a planar st-digraph G such that $v \neq s$ and $v \neq t$. There exists a path $P \subset G$ such that P is directed from s to t and P includes v.

Property 2. The external face of a planar st-digraph consists of edge (s,t) and of a directed path from s to t.

Let G be a planar st-digraph. For each edge $e = (u,v)$ of G, we denote by *left(e)* (resp. *right(e)*) the face to the left (resp. right) of e in G. Let s^* be the face $right((s,t))$, and let t^* be the face $left((s,t))$. In the rest of the paper we shall always assume that t^* is the external face of G. Faces s^* and t^* are highlighted in Figure 1(a).

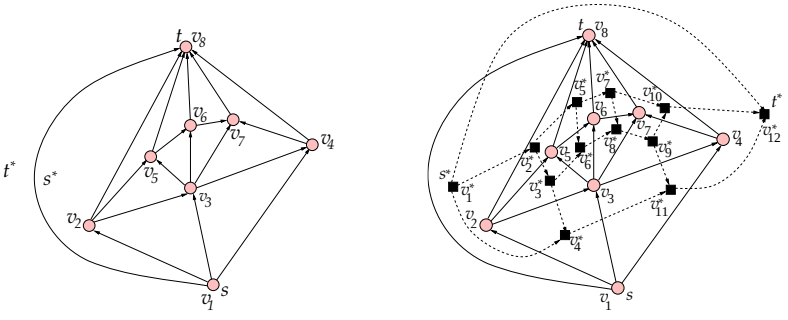


Fig. 1. (a) A planar st-digraph G with an st-numbering of its vertices. Digraph G is a maximal planar st-digraph. (b) st-digraph G (solid) and its dual st-digraph (dashed). The vertices of the dual st-digraph are numbered according to an st-numbering.

Let G be a planar st-digraph. The *dual* of G is the digraph denoted as G^* such that: (i) there is a vertex in G^* for each face of G; (ii) for every edge $e \neq (s,t)$ of G, G^* has an edge $e^* = (f,g)$ where $f = left(e)$ and $g = right(e)$; (iii) G^* has an edge (s^*,t^*). Figure 1(b) depicts with dashed edges the dual digraph of the digraph of Figure 1(a).

Property 3. Let G be a planar st-digraph and let G^* be the dual digraph of G. Graph G^* is a planar st-digraph with source s^* and sink t^*.

A planar st-digraph is said to be *maximal* if all its faces are triangles, i.e. the boundary of each face has exactly three vertices and three edges. Given any planar st-digraph G, one can always add edges that split faces of G in order to obtain a maximal planar st-digraph that includes G. Figure 1(a) is an example of a maximal planar st-digraph.

Property 4. Let G be a maximal planar st-digraph with more than three vertices. The dual of G is a planar st-digraph without multiple edges.

A planar drawing of a digraph is *upward* if all of its edges are curves monotonically increasing in a common direction which is called the *upward direction* of the drawing. For example, upward directions of an upward planar drawing could be the positive y-direction or the positive x-direction. Figure 1(a) is an example of an upward planar drawing. A digraph that admits an upward planar drawing is said to be *upward planar*. As proved in [7,24], upward planar digraphs are exactly the subgraphs of planar *st*-digraphs. Also, an upward planar digraph G can always be augmented to become a maximal planar *st*-digraph. This can be done by adding extra edges that "saturate" the faces of an upward planar drawing of G and by inserting at most two vertices on the external face of such upward planar drawing of G. One of these two extra vertices is the source of the external face of the drawing and the second one is the sink of the external face of the drawing. By using results of [7,12,24] the following can be proved.

Lemma 1. *Let G be an upward planar digraph with n vertices. There exists a maximal planar st-digraph with at most $n+2$ vertices that includes G. Also if an upward planar drawing of G is given, such an st-digraph can be computed in $O(n)$ time.*

An *st-numbering* of an *st*-digraph G with n vertices, is a numbering of its vertices with the integers $1, \ldots, n$ such that: (i) No two vertices have the same number; (ii) For every edge (u, v), the number of u is less than the number of v. For example, the indices of the vertices in Figure 1(a) are given according to an *st*-numbering of the depicted *st*-digraph. The number associated to a vertex v in an *st*-numbering of an *st*-digraph is called the *st-number* of v. Let u and v be two vertices of an *st*-digraph with a given *st*-numbering; if the *st*-number of u is less than the *st*-number of v we say that u *precedes* v and we denote it as $u <_{st} v$.

Lemma 2. [2] *Let G be a planar st-digraph with n vertices. An st-numbering of G can be computed in $O(n)$ time.*

3 Computing Upward Topological Book Embeddings

A 2-*page book* consists of a single vertical line, called *spine*, and of 2 half-planes called *pages* that share the spine as a common boundary. The half-plane on the left-hand side of the spine is the *left page*, the other one is the *right page*. Let p and q be two points of the spine. We say that p is *below* q and denote it as $p < q$ if the y-coordinate of p is smaller than the y-coordinate of q. Let p and q be two points of the spine of a 2-page book such that $p < q$. An *upward arc* (p, q) is a circular arc contained in one of the pages and passing through p, q and r, where r is a point of the perpendicular bisector of segment $\overline{pq}$ at a distance $\frac{d(p,q)}{2}$ from the spine. Points p and q are the *endpoints* of (p, q).

Let G be an upward planar digraph. An *upward topological book embedding* of G is an upward planar drawing Γ of G in a 2-page book such that: (i) All vertices of G are represented as points of the spine (the spine will also be called

spine of Γ); (ii) Each edge (u, v) of G is represented in Γ as either an upward arc or it consists of two upward arcs (u, z) and (z, v) such that (u, z) is in the left page and (z, v) is in the right page. Let $e = (u, v)$ be an edge of G represented in Γ by two upward arcs (u, z) and (z, v); we say that z is the *spine crossing* of e in Γ. Figure 2(a) shows an upward topological book embedding of the *st*-digraph depicted in Figure 1(a). We remark that, by definition, in an upward topological book embedding every edge can cross the spine at most once.

In the next subsections we study the problem of computing an upward topological book embedding of an upward planar digraph G. Based on Lemma 1, we will describe the drawing procedure by assuming that the input digraph is a maximal planar *st*-digraph. Subsection 3.1 introduces the notion of *k-facial subgraph* of an *st*-digraph, which is used as a guideline for the drawing procedure described in Subsection 3.2.

3.1 The *k*-Facial Subgraph

Let G be a maximal planar *st*-digraph with more than three vertices and let G^* be the dual digraph of G. By Property 4, G^* is a planar *st*-digraph without multiple edges; by Lemma 2, its vertices can be numbered according to an *st*-numbering. Hence, let $\{v_1^* = s^*, v_2^*, \ldots, v_m^* = t^*\}$ be the set of vertices of G^* where the indices are given according to an *st*-numbering of G^*. See, for example, Figure 1(b), where the vertices of the dual are numbered according to an *st*-numbering. By definition of dual *st*-digraph, a vertex v_i^* of G^* $(1 \leq i \leq m)$ corresponds to a face of G; in the remainder of the paper we shall denote as v_i^*

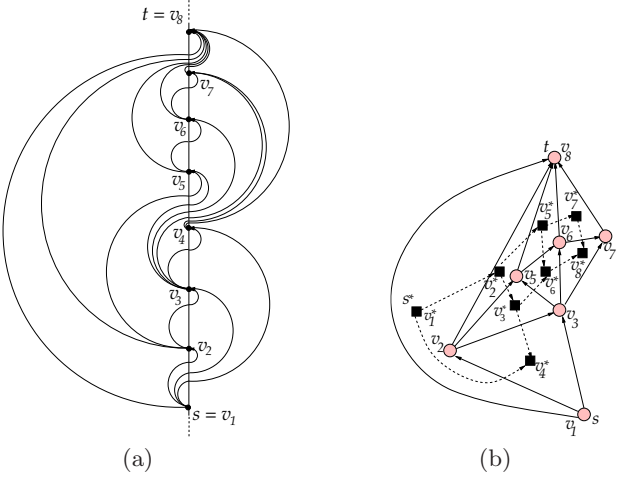


Fig. 2. (a) An upward topological book embedding of the maximal planar *st*-digraph of Figure 1(a). The drawing is computed by using Algorithm `Upward Spine Drawer` of Section 3.2. (b) The 8-facial subgraph of the maximal planar *st*-digraph of Figure 1(a).

both the vertex of the dual digraph G^* and its corresponding face in the primal digraph G.

Let V_k be the subset of the vertices of G that belong to faces $v_1^*, v_2^*, \ldots, v_k^*$. The subgraph of G induced by the vertices in V_k is called the *k-facial subgraph* of G and is denoted as G_k. Face v_k^* is called the *k-th face* of G. Observe that the topology of a k-facial subgraph of G depends on the particular st-numbering chosen for G^*. The drawing algorithm of the next section considers a sequence of k-facial subgraphs of G all defined on a same st-numbering of G^*. Hence, from now on we shall assume that G^* is given together with an st-numbering. As an example, Figure 2(b) shows the 8-facial subgraph of the maximal planar st-digraph depicted in Figure 1(a) assuming that the st-numbering of its dual is the one shown in Figure 1(b). The proof of Lemma 3 relies on properties of the st-numbering of G and of its dual.

Lemma 3. *Let G be a maximal planar st-digraph with m faces, let G_{k-1} be the $(k-1)$-facial subgraph of G $(2 \leq k \leq m)$ and let G_k be the k-facial subgraph of G. Let v_k^* be the k-th face of G consisting of edges (w, w'), (w', w''), and (w, w''). One of the following statements holds:*

- (S_1)**:** *(w, w'') is an edge of the external face of G_{k-1}; (w, w') and (w', w'') are edges of the external face of G_k.*
- (S_2)**:** *(w, w') and (w', w'') are edges of the external face of G_{k-1}; (w, w'') is an edge of the external face of G_k.*

The following lemma can be proved by induction and by means of Lemma 3.

Lemma 4. *Let G be a maximal planar st-digraph with m faces and let G_k be the k-facial subgraph of G $(1 \leq k \leq m)$. G_k is a planar st-digraph.*

3.2 The Upward Spine Drawer Algorithm

Let G be a maximal planar st-digraph with m faces, and let $v_1 = s, \ldots, v_n = t$ be the vertices of G ordered according to an st-numbering of G. Algorithm `Upward Spine Drawer` receives G as input and it computes as output an upward topological book embedding of G. The computed upward topological book embedding respects the given upward planar embedding for G. In order to properly describe the algorithm, we need two additional definitions. Let Γ be an upward topological book embedding and let p be a point on the spine of Γ. We say that p is *visible from the right-hand side* if the horizontal line through p does not intersect any upward arc of Γ in the right page. Let v be a vertex of Γ and let p be a point of the spine such that $v < p$. We say that segment $\overline{pv}$ is a *safe interval of* v if: (i) Every point of $\overline{pv}$ is visible from the right-hand side and (ii) $\overline{pv}$ does not contain endpoints of any upward arcs (either in the left or in the right page). Note that the safe interval of v is assumed to be an open set.

A high-level description of Algorithm `Upward Spine Drawer` is as follows. The algorithm computes an upward topological book embedding of G on a 2-page book in m steps. At Step 1, it computes an upward topological book embedding

of the 1-facial subgraph G_1. Let Γ_{k-1} be the drawing computed at the end of Step $(k-1)$ $(2 \leq k \leq m)$. At Step k, a drawing Γ_k of the k-facial subgraph G_k is computed by adding a new face to the drawing Γ_{k-1} of G_{k-1}. At each step the following invariant properties are maintained.

I1: Let w and w' be two vertices of the external face of G_k such that $w <_{st} w'$ in the st-numbering of G. Then $w < w'$ in Γ_k.
I2: For each vertex w of the external face of G_k, w is visible from the right-hand side and w has a safe interval.

A more detailed description of the steps executed by Algorithm `Upward Spine Drawer` is given below; Λ denotes the spine of the 2-page book.

- Step 1, computation of Γ_1: Let $\{s, t, w\}$ be the vertices of the boundary of face v_1^*. Draw s and t along Λ such that s is below t. Let z be a point of the spine such that $s < z < t$. Let (s, z) be the upward arc from s to z in the left page and let (z, t) be the upward arc from z to t in the right page. Draw edge (s, t) in Γ_1 as the curve formed by (s, z) followed by (z, t). Represent w as point of the spine such that $s < w < z$. Select two points z_s and z_w of the spine such that $s < z_s < w$ and $w < z_w < z$. Edge (s, w) is drawn as two upward arcs (s, z_s), (z_s, w) into the left and right page, respectively. Edge (w, t) is drawn as two upward arcs (w, z_w), (z_w, t), into the left and right page, respectively.
- Step k, computation of Γ_k $(2 \leq k \leq m)$: Let Γ_{k-1} be the drawing of G_{k-1} and let $w_1 = s, w_2, \ldots, w_h = t$ be the counterclockwise sequence of the vertices along the external face of Γ_{k-1}. Add face v_k^* to Γ_{k-1} as follows.
  - Statement S_1 of Lemma 3 holds. The boundary of face v_k^* is a three cycle having two consecutive vertices of the external face of Γ_{k-1}, say w_i and w_{i+1} $(1 \leq i \leq h-1)$, and a vertex v of the external face of G_k. Let p be a point above w_i such that segment $\overline{w_i p}$ is the safe interval of w_i. Draw v as a point in the safe interval of w_i. Let z_{w_i} be a point of Λ such that $w_i < z_{w_i} < v$. Draw edge (w_i, v) as the upward arc (w_i, z_{w_i}) in the left page followed by the upward arc (z_{w_i}, v) in the right page. Let z_v be a point of Λ such that $v < z_v < p$. Draw edge (v, w_{i+1}) as the upward arc (v, z_v) in the left page followed by the upward arc (z_v, w_{i+1}) in the right page.
  - Statement S_2 of Lemma 3 holds. The boundary of face v_k^* is a three cycle having three consecutive vertices of the external face of Γ_{k-1} denoted as w_i, w_{i+1}, and w_{i+2} $(1 \leq i \leq h-2)$. Drawing Γ_k is computed by adding edge (w_i, w_{i+2}) to Γ_{k-1} as follows. Let z_{w_i} be a point in the safe interval of w_i. Draw (w_i, w_{i+2}) as the upward arc (w_i, z_{w_i}) in the left page followed by the upward arc (z_{w_i}, w_{i+2}) in the right page.

Figure 2(a) is an example of drawing computed by Algorithm `Upward Spine Drawer` when the input is the maximal planar st-digraph of Figure 1(a).

Lemma 5. *Let G be a maximal planar st-digraph. Algorithm* `Upward Spine Drawer` *maintains Invariants* `I1` *and* `I2`.

Lemma 6. *Let G be a maximal planar st-digraph. Algorithm* `Upward Spine Drawer` *computes an upward topological book embedding of G.*

We are now ready to present the main result of this section.

Theorem 1. *Every upward planar digraph G with n vertices admits an upward topological book embedding. Also, if an upward planar drawing of G is given, such upward topological book embedding can be computed in $O(n)$ time.*

In the next two sections we discuss applications of Theorem 1 to problems of graph drawing and computational geometry. Namely, Section 4 is devoted to upward drawings with constraints on the position of the vertices, while Section 5 is concerned with simultaneous embeddings of pairs of upward planar digraphs sharing their vertex set.

4 Upward Point-Set Embeddings

Let S be a set of n distinct points on the plane and let G be an upward planar digraph with n vertices. An *h-bend upward point-set embedding* of G on S is an upward planar drawing of G such that each vertex is mapped to a distinct point of S and every edge has at most h bends (notice that the mapping of the vertices to the points of S is not part of the input). A digraph G is *h-bend upward point-set embeddable* if it has an h-bend upward point-set embedding on *any* set of n points in the plane. In [9] it has been proved that an upward planar digraph is 1-bend upward point-set embeddable if and only if it has an upward topological book embedding such that no edge crosses the spine (i.e. it has an upward book embedding on two pages). It has also been proved that the following classes of digraphs admit this type of drawing: tree dags [21], unicyclic dags [21], and two-terminal series-parallel digraphs [9]. Hence, all graphs in these families are 1-bend upward point-set embeddable. However, not all upward planar digraphs have an upward topological book embedding without spine crossings [21] and therefore at least two bends are necessary in the general case. By using Theorem 1 and techniques from [10,22] we can prove that two bends per edge are actually always sufficient. In the following theorem the area of a drawing is the area of the smallest axis-aligned rectangle enclosing the drawing.

Theorem 2. *Every upward planar digraph G with n vertices admits a 2-bend upward point-set embedding on any set S of n distinct points in the plane. Also, if an upward planar drawing of G is given, such 2-bend upward point-set embedding can be computed in $O(n \log n)$ time and in area $O(W^3)$, where W is the width of the smallest axis-aligned rectangle enclosing S.*

5 Upward Simultaneous Embeddings

Let G_1 and G_2 be two planar graphs with the same vertex set, i.e. $V(G_1) = V(G_2) = V$. A *simultaneous embedding* of G_1 and G_2 is a pair of drawings of

G_1 and G_2 such that each drawing is planar and each vertex is represented by the same point in both drawings. The problem of computing a simultaneous embedding of two undirected planar graphs is a classical subject of investigation in the graph drawing literature (see, e.g. [5,6,11,14,16]). This section considers the upward version of this problem and uses Theorem 1 together with techniques from [11,14] to establish upper bounds on the area and number of bends per edge of the computed drawings.

Let G_1 and G_2 be two upward planar digraphs with the same vertex set, i.e. $V(G_1) = V(G_2) = V$. An *upward simultaneous embedding* of G_1 and G_2 is a pair of upward planar drawings Γ_1 of G_1 and Γ_2 of G_2 such that each vertex is represented by the same point in both drawings. An upward simultaneous embedding of G_1 and G_2 will also be denoted as $< \Gamma_1, \Gamma_2 >$. Note that the upward directions of Γ_1 and Γ_2 in $< \Gamma_1, \Gamma_2 >$ are not required to be the same.

Theorem 3. *Every pair of upward planar digraphs G_1 and G_2 such that $V(G_1) = V(G_2) = V$ admits an upward simultaneous embedding with at most two bends per edge. Also, if a pair of upward planar drawings of G_1 and G_2 is given, such upward simultaneous embedding can be computed in $O(n)$ time and in area $O(n^2) \times O(n^2)$, where $n = |V|$.*

6 Conclusions and Open Problems

In this paper we presented a unified approach to studying book-, point-set, and simultaneous embeddability problems of upward planar digraphs. The approach is based on a linear time strategy to compute an upward planar drawing of an upward planar digraph such that all vertices are collinear and each edge has at most two bends. Besides having impact in relevant application domains of graph drawing and computational geometry, the presented results open new research directions in the area of upward planarity with constraints of the positions of the vertices. We therefore conclude this paper by discussing some of the most interesting questions that can be inspired by the presented results.

Upward book embeddability: Theorem 1 shows that an upward topological book embedding of an upward planar digraph can be computed such that every edge crosses the spine at most once. It would be interesting to study the problem of computing upward topological book embeddings with the minimum number of spine crossings.

Upward point-set embeddability: Theorem 2 shows that every upward planar digraph with n vertices has a 2-bend upward point-set embedding on any set on n distinct points in the plane. In [22] it is proved that point-set embeddings of undirected planar graphs may require two bends per edge. This immediately implies that the same lower bound also applies to the upward planar case, and therefore the statement of Theorem 2 is tight in terms of bends per edge. However, it is well-known that every (undirected) outerplanar graph with n vertices has a point-set embedding on any set of n points in general position with straight-line edges and that the outerplanar graphs are the largest family of graphs with

this property [18]. It would be interesting to characterize those upward planar digraphs that have an upward point-set embedding with straight-line edges on any set of points in general position.

Upward simultaneous embeddability: Theorem 3 shows that any two upward planar digraphs have an upward simultaneous embedding with at most two bends per edge. It would be interesting to understand whether the number of bends per edge stated in Theorem 3 is also necessary in some cases. We recall that one bend on some of the edges may be required to simultaneously embed pairs of undirected planar graphs [5,14,17] and hence the same lower bound also applies to the problem of computing upward simultaneous embeddings.

A related question asks whether a straight-line upward simultaneous embedding of two upward planar digraphs G_1 and G_2 is always possible in the *no-mapping scenario*. In this scenario, the goal is to compute a pair $< \Gamma_1, \Gamma_2 >$ of straight-line upward planar drawings of G_1 and of G_2 such that the set of points representing the vertices is the same in Γ_1 and in Γ_2, but each vertex can have different coordinates in the two drawings. For example, a straight-forward consequence of the literature is that any number of tree dags and of unicyclic dags can be upward simultaneously embedded without mapping and with straight-line edges. Namely, in [21] it is proved that these graphs admit an upward book embedding with all edges in the same page. Thus, choose a set S of n points in general position such that: (i) the points are in convex position, (ii) all points have distinct y-coordinates, and (iii) the two extreme points in the y-direction are adjacent in the convex hull and all the remaining points are to the left of the upward-directed line they define. Now compute a straight-line upward point-set embedding of each tree or unicyclic dag with n vertices by mapping the vertices to the points of S by increasing y-coordinate and according to the below-to-above order of these vertices along the spine. We find it interesting to study the general question about whether any pair of upward planar digraphs (not just tree dags or unicyclic dags) admit an upward simultaneous embedding without mapping.

References

1. Alzohairi, M., Rival, I.: Series-parallel ordered sets have pagenumber two. In: North, S. (ed.) GD 1996. LNCS, vol. 1190, pp. 11–24. Springer, Heidelberg (1997)
2. Battista, G.D., Eades, P., Tamassia, R., Tollis, I.: Graph Drawing. Prentice-Hall, Englewood Cliffs (1999)
3. Bose, P.: On embedding an outer-planar graph in a point set. Comput. Geom. 23(3), 303–312 (2002)
4. Bose, P., McAllister, M., Snoeyink, J.: Optimal algorithms to embed trees in a point set. Journal of Graph Algorithms and Applications 2(1), 1–15 (1997)
5. Brass, P., Cenek, E., Duncan, C., Efrat, A., Erten, C., Ismailescu, D., Kobourov, S., Lubiw, A., Mitchell, J.: On simultaneous planar graph embeddings. Computational Geometry. Theory and Applications 36(2), 117–130 (2007)
6. Cappos, J., Estrella-Balderrama, A., Fowler, J., Kobourov, S.G.: Simultaneous graph embedding with bends and circular arcs. In: Kaufmann, M., Wagner, D. (eds.) GD 2006. LNCS, vol. 4372, pp. 95–107. Springer, Heidelberg (2007)

7. Di Battista, G., Tamassia, R.: Algorithms for plane representations of acyclic digraphs. Theor. Comput. Sci. 61(2-3), 175–198 (1988)
8. Di Battista, G., Tamassia, R., Tollis, I.G.: Constrained visibility representations of graphs. Inf. Process. Lett. 41(1), 1–7 (1992)
9. Di Giacomo, E., Didimo, W., Liotta, G., Wismath, S.: Book embeddability of series-parallel digraphs. Algorithmica 45(4), 531–547 (2006)
10. Di Giacomo, E., Didimo, W., Liotta, G., Wismath, S.K.: Curve-constrained drawings of planar graphs. Comput. Geom. Theory Appl. 30(1), 1–23 (2005)
11. Di Giacomo, E., Liotta, G.: Simultaneous embedding of outerplanar graphs, paths, and cycles. International Journal of Computational Geometry and Applications 17(2), 139–160 (2007)
12. Didimo, W.: Upward planar drawings and switch-regularity heuristics. Journal of Graph Algorithms and Applications 10(2), 259–285 (2006)
13. Enomoto, H., Miyauchi, M., Ota, K.: Lower bounds for the number of edge-crossings over the spine in a topological book embedding of a graph. Discrete Applied Mathematics 92, 149–155 (1999)
14. Erten, C., Kobourov, S.: Simultaneous embedding of planar graphs with few bends. Journal of Graph Algorithms and Applications 9(3), 347–364 (2005)
15. Giordano, F., Liotta, G., Mchedlidze, T., Symvonis, T.: Computing upward topological book embeddings of upward planar digraphs. Technical report, RT-007-02 (2007)
16. Frati, F.: Embedding graphs simultaneously with fixed edges. In: Kaufmann, M., Wagner, D. (eds.) GD 2006. LNCS, vol. 4372, pp. 108–113. Springer, Heidelberg (2007)
17. Geyer, M., Kaufmann, M., Vrto, I.: Two trees which are self-intersecting when drawn simultaneously. In: Healy, P., Nikolov, N.S. (eds.) GD 2005. LNCS, vol. 3843, pp. 201–210. Springer, Heidelberg (2006)
18. Gritzmann, P., Mohar, B., Pach, J., Pollack, R.: Embedding a planar triangulation with vertices at specified points. Amer. Math. Monthly 98(2), 165–166 (1991)
19. Heath, L.S., Pemmaraju, S.V.: Stack and queue layouts of posets. SIAM Journal on Discrete Mathematics 10, 599–625 (1997)
20. Heath, L.S., Pemmaraju, S.V.: Stack and queue layouts of directed acyclic graphs: Part II. SIAM Journal on Computing 28, 1588–1626 (1999)
21. Heath, L.S., Pemmaraju, S.V., Trenk, A.: Stack and queue layouts of directed acyclic graphs: Part I. SIAM Journal on Computing 28, 1510–1539 (1999)
22. Kaufmann, M., Wiese, R.: Embedding vertices at points: Few bends suffice for planar graphs. Journal of Graph Algorithms and Applications 6(1), 115–129 (2002)
23. Kaufmann, M., Wagner, D. (eds.): Drawing Graphs. LNCS, vol. 2025. Springer, Heidelberg (2001)
24. Kelly, D.: Fundamentals of planar ordered sets. Discrete Math. 63, 197–216 (1987)
25. Nishizeki, T., Rahman, M.S.: Planar Graph Drawing. Lecture Notes Series on Computing, vol. 12. World Scientific (2004)
26. Nowakowski, R., Parker, A.: Ordered sets, pagenumbers and planarity. Order 6, 209–218 (1989)
27. Sugiyama, K.: Graph Drawing and Application for Software amd Knowledge Engineers. World Scientific, Singapore (2001)
28. Yannakakis, M.: Embedding planar graphs in four pages. Journal of Computer and System Sciences 38, 36–67 (1989)

Algorithms for the Hypergraph and the Minor Crossing Number Problems

Markus Chimani and Carsten Gutwenger

Department of Computer Science, University of Dortmund, Germany
{markus.chimani,carsten.gutwenger}@cs.uni-dortmund.de

Abstract. We consider the problems of hypergraph and minor crossing minimization, and point out a relation between these two problems which has not been exploited before. We present some complexity results regarding the corresponding edge and node insertion problems. Based on these results, we give the first embedding-based heuristics to tackle both problems and present a short experimental study. Furthermore, we give the first exact ILP formulation for both problems.

1 Introduction

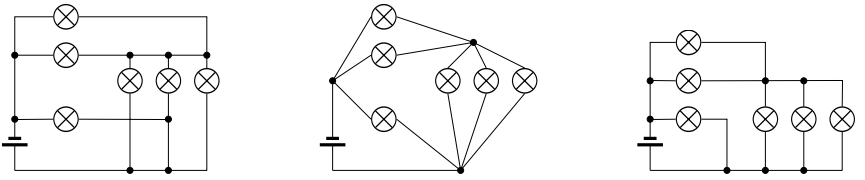
A *drawing* of a graph G on the plane is a one-to-one mapping of each vertex to a point in $\mathbb{R}^2$, and each edge to a curve between its two endpoints. The curve is not allowed to contain other vertices than its two endpoints. A *crossing* is a common point of two curves, other than their endpoints. The *crossing number* $\mathrm{cr}(G)$ then is the smallest number of crossings in any drawing of G. The corresponding crossing minimization (CM) problem has been widely studied in the literature, see [15] for an extensive bibliography.

In this paper we consider the problem of finding the *Hypergraph Crossing Number*, as defined in the following section. We do so by exploiting a connection between this crossing number and the *Minor Crossing Number*, i.e., the smallest crossing number of any graph G' which has G as its minor, denoted by $G \preceq G'$. Especially the latter concept has yet only been studied in the context of theoretical lower and upper bounds [2,3], but was never before tackled algorithmically.

Besides from their theoretical appeal, these problems also occur, e.g., for crossing minimal layouts of electrical wiring schemes [3], cf. Fig. 1. Usually, the exact topology of such a wiring scheme G'' is not interesting for the connected subgraphs which have the same electric potential. Hence we can "merge" these nodes into one node, which is exactly the operation to obtain a minor G, compute the minor crossing number $\mathrm{mcr}(G)$ and expand the graph accordingly to obtain G'. In this example, we can observe the connection to hypergraphs: by seeing the impedances on the wires as nodes, we can interpret the wires on the same potential as *hyperedges*, i.e., edges with multiple incident nodes.

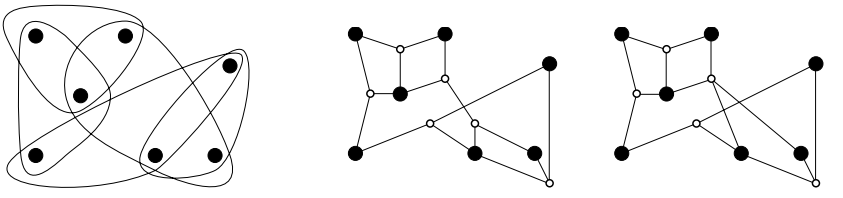
The computation of both crossing numbers is NP-complete. In this context, we investigate several subproblems of edge and node insertion (Sections 3 and 5), and establish novel heuristics for both kinds of crossing numbers (Section 4). We

T. Tokuyama (Ed.): ISAAC 2007, LNCS 4835, pp. 184–195, 2007.



(a) (b) (c)

Fig. 1. The wiring scheme (a) cannot be drawn without any crossing. By computing a minor (b) and considering a realizing graph (c), we obtain an equivalent but planar wiring scheme.



(a) Subset (b) Edge, tree-based (c) Edge, point-based

Fig. 2. A hypergraph, drawn using different drawing styles

also implemented these algorithms and give an experimental study in Section 7. Furthermore, in Section 6 we sketch the first exact ILP formulation able to provably solve both problems to optimality.

The next subsections will give the detailed definitions of the considered problems and summarize our results.

1.1 Definitions

A hypergraph $H = (V, \mathcal{F})$ differs from an ordinary graph that instead of edges—which have exactly two incident nodes—we consider *hyperedges*: a hyperedge $F \in \mathcal{F}$ is a proper subset of V, i.e., $F \subset V$, with $|F| \geq 2$. See, e.g., [10] for details. There are two major variants on how to draw hypergraphs [12], cf. Fig. 2: the *subset-standard* and the *edge-standard*. The first variant becomes very confusing with more hyperedges, and it is ambiguous how to define a consistent notion of crossings. Hence, most applications, cf. [5,11,14], focus on the edge-standard which allows two subvariants: in the *tree-based* drawing style, each hyperedge F is drawn as a tree-like structure of lines, whose leaves are the incident nodes of F. If we restrict the tree-like structure of every hyperedge to be a star, we obtain the *point-based* drawing style.

Formally, each hyperedge $F \in \mathcal{F}$ has a set of associated *hypernodes* N_F, which form the branching points of the line tree: each node $v \in F$ is connected to exactly one $n \in N_F$, and all hypernodes N_F are treewise connected. By this *tree-based transformation* we obtain a traditional graph L. We denote the set of all graphs L obtainable by such transformations by $\mathcal{L}(H)$, and can naturally define:

Definition 1 (Tree-based Hypergraph Crossing Number). Let H be a hypergraph. We define the *tree-based hypergraph crossing number* as

$$\mathrm{thcr}(H) := \min_{L \in \mathcal{L}(H)} \mathrm{cr}(L).$$

The tree-based hypergraph crossing number has the elegant property that it is equivalent to the traditional crossing number if all hyperedges have cardinality 2. Because of this property, computing $\mathrm{thcr}(H)$ is NP-complete.

We further define the *point-based transformation* $\Lambda(H)$ as the special tree-based transformation, where each hyperedge has exactly one associated hypernode, i.e., $\Lambda(H) := (V \cup \mathcal{F}, E(H))$ with $E(H) := \{(v, F) \mid v \in F, F \in \mathcal{F}\}$. Clearly, this leads to the point-based drawing style and the definition of the *point-based hypergraph crossing number* $\mathrm{phcr}(H) := \mathrm{cr}(\Lambda(H))$.

Observation 1. For any $L \in \mathcal{L}(H)$ we have $\Lambda(H) \preceq L$, i.e., the point-based transformation of H is the minor of any tree-based transformation of H.

Point-based hypergraph planarity of H can be defined as $\mathrm{phcr}(H) = 0$ straightforwardly and is equivalent to Zykov planarity [10]. It can be efficiently tested by transforming H into $\Lambda(H)$ in linear time and applying any traditional linear-time planarity testing algorithm to $\Lambda(H)$. Analogously, *tree-based hypergraph planarity* can be defined as $\mathrm{thcr}(H) = 0$. Since $L \in \mathcal{L}(H)$ is planar if and only if $\Lambda(H)$ is planar, all three planarity definitions are equivalent.

Obviously, the point-based hypergraph crossing minimization of H is equivalent to the traditional crossing minimization on the graph $\Lambda(H)$. Hence we will focus on computing $\mathrm{thcr}(H)$. To understand this crossing number better, we first have to focus on the *minor crossing number* [3], also known as the *minor-monotone crossing number*, for traditional graphs:

Definition 2 (Minor Crossing Minimization Problem). Let $G = (V, E)$ be an undirected graph. The *Minor Crossing Minimization Problem* (MCM) is to find a *realizing graph* $G' = (V', E') \succeq G$ with $\mathrm{cr}(G') = \mathrm{mcr}(G) := \min_{G'' \succeq G} cr(G'')$, i.e., the minor crossing number of G.

Let $W \subseteq V$. We can define a minor relation $G' \succeq_W G$, i.e., G is a W-minor of G' if we can obtain G' from G by only expanding nodes of W. This leads to the more general *W-restricted Minor Crossing Number* $\mathrm{mcr}_W(G)$, i.e., the smallest crossing number of any graph $G' \succeq_W G$, and the *W-restricted Minor Crossing Minimization Problem* (RMCM). Clearly $\mathrm{mcr}_W(G) = \mathrm{mcr}(G)$, if $W = V$. Since nodes with degree less than 4 are irrelevant for the differences between the traditional and the minor crossing number, we have:

Proposition 1. *Let* $\hat{\mathcal{F}} := \{F \in \mathcal{F} \mid |F| \geq 4\}$. *The tree-based hypergraph crossing number is equivalent to the* $\hat{\mathcal{F}}$*-restricted minor crossing number of* $\Lambda(H)$*, i.e.,* $\mathrm{thcr}(H) = \mathrm{mcr}_{\hat{\mathcal{F}}}(\Lambda(H))$.

Hence we have to find a realizing graph $\Lambda' \succeq_{\mathcal{F}'} \Lambda(H)$ with smallest crossing number, i.e., we may obtain Λ' only by expanding hypernodes of degree at least 4.

1.2 Edge and Node Insertion Results

The MCM problem was first stated in [3] and was shown to be NP-complete in [9]. In Section 2, we will discuss two alternative viewpoints of MCM, which are fundamental for the results presented thereafter. In the following we will always consider the W-restricted variant of the minor crossing number. Since W may be the full set V, the results clearly also hold for the traditional minor-monotonous case. Note that we consider two embeddings, or rotation-systems, Γ of G and Γ' of G' ($G \preceq G'$) *equivalent*, if we can obtain Γ by performing the necessary minor operations stepwise on G' and Γ' in the natural way: let us merge the connected nodes a and b with their respective cyclic orders $\pi_a = \langle \{a, b\}, e_1, \ldots, e_{\deg(a)-1} \rangle$ and $\pi_b = \langle \{b, a\}, f_1, \ldots, f_{\deg(b)-1} \rangle$ of their incident edges. The new node c will have the cyclic order $\pi_c = \langle e_1, \ldots, e_{\deg(a)-1}, f_1, \ldots, f_{\deg(b)-1} \rangle$.

Similar to the corresponding problems for the traditional crossing number, we can state the following related problems:

Definition 3 (Minor Edge Insertion). Let $G = (V, E)$ be a planar undirected graph, and let $e = \{s, t\} \in V \times V \setminus E$ be an edge not yet in G. The *W-restricted Minor Edge Insertion Problem with Variable Embedding (MEIV)* is to find the W-restricted minor crossing number of the graph $G + e$, under the restriction that the realizing drawing induces a planar drawing of G.

Given a specific planar embedding Γ of G, the *W-restricted Minor Edge Insertion Problem with Fixed Embedding (MEIF)* is to find the W-restricted minor crossing number of the graph $G + e$, under the restriction that the realizing drawing induces an embedding of G equivalent to Γ.

For both problems, the equivalent problems concerning the traditional crossing number can be solved in linear time. In Section 3, we show:

Theorem 1. *MEIF and MEIV can be solved optimally in linear time.*

In Section 4, we show how to use this result to obtain a heuristic following the planarization approach.

Definition 4 (Minor Node Insertion). Let $G = (V, E)$ be a planar undirected graph, let $v \notin V$ be a node not yet in G, and let E' be edges connecting v with nodes of V. Let $W^- := W$ and $W^+ := W \cup \{v\}$. The *W^--restricted (W^+-restricted) Minor Node Insertion Problem with Variable Embedding* is to find the W^--restricted (W^+-restricted) minor crossing number of the graph $G' = (V \cup \{v\}, E \cup E')$, under the restriction that the realizing drawing induces a planar drawing of G. We abbreviate these problems MNIV$^-$ and MNIV$^+$, respectively.

Given a specific planar embedding Γ of G, the *W^--restricted (W^+-restricted) Minor Node Insertion Problem with Fixed Embedding* is to find the W^--restricted (W^+-restricted) minor crossing number of the graph G', under the restriction that the realizing drawing induces an embedding of G equivalent to Γ. We abbreviate these problems MNIF$^-$ and MNIF$^+$, respectively.

The equivalent problem for the traditional crossing number and a fixed embedding can be solved in $\mathcal{O}(|V| \cdot |E'|)$ time. An analogous algorithm, together with the ideas of Theorem 1, can be used to show (see Section 5):

Theorem 2. *MNIF$^-$ is solvable in $\mathcal{O}(|V| \cdot |E'|)$ time.*

The problem for the traditional crossing number where all embeddings are considered, and therefore a special case of MNIV$^-$, is still an open problem. In contrast to these results, we can show that the problem is hard when the inserted node is allowed to be expanded:

Theorem 3. *MNIF$^+$ and MNIV$^+$ are NP-complete. This also holds for the case $W = V$, i.e., non-restricted minor-monotonicity.*

Corollary 1. *Let $H = (V, \mathcal{F})$ be a hypergraph, and $F \notin \mathcal{F}$ a hyperedge not yet in H. Under the restriction that H has to be drawn planar and independent on whether a specific embedding of H is given or not, we have: computing* $\operatorname{thcr}(H + F)$ *is NP-complete.*

The theorem is based on the observation that we can turn any planar Steiner tree problem instance (NP-complete, [6]) into a corresponding MNIF$^+$ problem, cf. Section 5. The corollary does not follow from Theorem 3 itself, but from its proof. Furthermore, since computing $\operatorname{thcr}(H)$ is a special case of a W-resticted minor crossing number we have in particular:

Observation 2. The heuristic and exact algorithms presented below can be used to solve the tree-based hypergraph crossing number problem heuristically and to optimality, respectively.

2 General Observations

In the following, we will always consider an undirected graph $G = (V, E)$ with $W \subseteq V$, and we are interested in $\operatorname{mcr}_W(G)$. For the following algorithms, there are two points of view which are helpful when discussing the problem of minor crossing numbers: we can replace each node $v \in W$ with $\deg(v) \geq 4$ by an *expansion tree* T_v, which is incident to all nodes—or their respective expansion trees—originally incident to v. The nodes of T_v are called the *split nodes* of v. The RMCM problem can then be reformulated as finding a *tree expansion* G', i.e., a graph obtained by such transformations, with smallest crossing number.

Another possibility to view the problem is that in the traditional crossing number problem, edges are allowed to cross. For the minor crossing number, edges are allowed to cross through vertices, and moreover vertices may even "cross" other vertices. Such crossings can be seen as crossings between an expansion tree and a traditional edge, or between two expansion trees, respectively.

3 Optimal Edge Insertion

In this Section, we present linear time algorithms for MEIF and MEIV. Our task is to find a tree expansion G' of G along with an insertion path connecting s and t, i.e., an ordered list of edges that are crossed when inserting e. Observe that it is never necessary to expand s or t.

Fixed Embedding. Let Γ be an embedding of G. We define a directed graph $D_{\Gamma,s,t} = (N, A)$ as follows. N contains a node n_f for each face $f \in \Gamma$ and a node n_v for each node $v \in W \cup \{s, t\}$. Each arc $a \in A$ has an associated cost $c_a \in \{0, 1\}$; we have the following arcs:

- For each pair f, f' of adjacent faces, we have two arcs $(n_f, n_{f'})$ and $(n_{f'}, n_f)$ with cost 1.
- For each node $v \in W \setminus \{s, t\}$ and face f incident to v we have an arc (n_v, n_f) with cost 1 and an arc (n_f, n_v) with cost 0.
- Finally, we have arcs (n_s, n_f) for each face f incident to s and (n_f, n_t) for each face f incident to t; all these arcs have cost 0.

Then, the solution to MEIF is the length of a shortest path p in $D_{\Gamma,s,t}$ from n_s to n_t; each arc $(n_f, n_{f'})$ in p corresponds to crossing an edge separating f and f', and each subpath $(n_f, n_v), (n_v, n_{f'})$ corresponds to splitting node v and crossing the edge resulting from the split.

The number of nodes in N is bounded by $|V| + |F|$, where F is the set of faces in Γ, and the number of arcs in A by $4 \cdot |E|$, since we have at most four arcs per edge. Hence, we can apply breadth-first-search for finding a shortest path in $D_{\Gamma,s,t}$ which takes time $\mathcal{O}(|V| + |E|)$. We remark that BFS can easily be extended to graphs with 0/1-arc costs. Thus, we can solve MEIF in linear time.

Variable Embedding. In order to solve MEIV, we adapt the algorithm by Gutwenger et al. [8] which solves the problem for the traditional crossing number, i.e., $W = \emptyset$ and no tree expansions are possible. They showed that it is sufficient to consider the shortest path $B_0, v_1, B_1, \ldots, v_k, B_k$ in the BC-tree of G and independently compute optimal edge insertion paths in the (biconnected) blocks B_i from v_i to v_{i+1} ($0 \leq i \leq k$, $v_0 = s$, and $v_{k+1} = t$). This is also true when we are allowed to split the nodes W: concatenating the respective paths in the blocks results in a valid insertion path, and alternately crossing edges from different blocks or splitting (and crossing through) a cut vertex v_i would result in unnecessary crossings.

Thus, we can restrict ourselves to a biconnected graph G. Let $\mathcal{T}$ be the *SPQR-tree* of G (an SPQR-tree represents the decomposition of a biconnected graph into its triconnected components, please refer to [8] for details). We consider the shortest path $p = \mu_1, \ldots, \mu_h$ in $\mathcal{T}$ from a node μ_1 whose skeleton contains s to a node μ_h whose skeleton contains t. Let S_i be the skeleton of μ_i ($1 \leq i \leq h$). The *representative* $\text{rep}(v)$ of a node $v \in G$ in a skeleton S_i is either v itself if $v \in S_i$, or the edge $e \in S_i$ whose expansion graph contains v. If $W = \emptyset$, the optimal

algorithm only considers the R-nodes—triconnected components with therefore unique embeddings—on p and independently computes optimal edge insertion paths in fixed embeddings of the respective skeletons from $\text{rep}(s)$ to $\text{rep}(t)$. If the representative is an edge, we assume that a virtual node is placed on this edge and serves as start or endpoint of the insertion path.

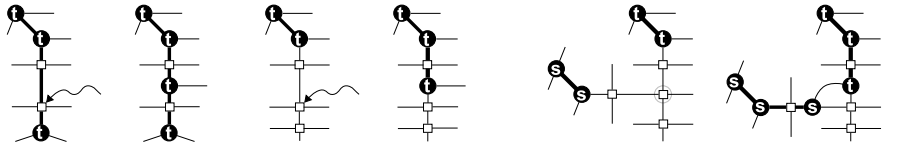
This approach is invalid if $W \neq \emptyset$: an optimal insertion path in a skeleton S_i might cross through an endpoint x of the edge representing t in S_i, and continuing this path from x in S_{i+1} might save a crossing. We circumvent this problem by processing p in order from μ_1 to μ_h: for each R-node μ_i where $\text{rep}(t)$ is an edge $e_t = (x, y)$, we compute three insertion paths p_x, p_y, and p_e in a fixed embedding of S_i, which are optimal insertion paths to x, y, and e_t, respectively. Observe that for the respective lengths ℓ_x, ℓ_y, and ℓ_e of these paths we have $\ell_e \leq \ell_x, \ell_y \leq \ell_e + 1$. If $x \in W$, $\ell_x = \ell_e$, and x is contained in the skeleton of the next processed R-node μ_j, then x is a possible start node for an insertion path in S_j; the analogous is true for y; $\text{rep}(s)$ is always a possible start node. We compute the optimal insertion paths in the R-node skeletons by slightly modifying the search network introduced for MEIF. We introduce a super start node s^* and connect it to the possible start nodes. Then we compute shortest paths from s^* to $\text{rep}(t)$ and, if this is an edge e_t, to the endpoints of e_t.

After processing all nodes on p, we reconstruct the optimal insertion path backwards from t to s. The insertion path in S_h ends in t; we determine which insertion path in the preceding R-node skeleton to chose by checking which start node is used, until we reach s. This algorithm can be implemented in linear time, thus showing that MEIV can be solved in linear time as well.

4 The Planarization Approach for RMCM

The planarization approach is a well-known and successful heuristic for traditional crossing minimization; see [7] for an experimental study. First, a planar subgraph is computed, and then the remaining edges are inserted one after another by computing edge insertion paths and inserting the edges accordingly, i.e., edge crossings are replaced by dummy vertices with degree 4.

In order to apply the algorithms from the previous section in a planarization approach for RMCM, we need to generalize them, since the insertion of edges splits nodes and thereby expands them to trees. Furthermore, edges of G and edges resulting from node splits get subdivided by dummy vertices during the course of the planarization. We call the resulting paths *edge paths* and *tree paths*, respectively. Hence, we are not simply given two nodes s and t but two node sets S and T, and we have to find an insertion path connecting a node of S with a node of T. Thereby, S (T) is the set of all split nodes of s (t) and all dummy nodes on edge or tree paths starting at a split node of s (t). The dummy nodes in these sets have the property that a simple extension of a tree expansion is sufficient to connect an insertion path to a correct split node; see Fig. 3 for a visual description.



(a) Dummy on target (b) Dummy near target (c) Dummy near source and target

Fig. 3. Modification of insertion paths ending at dummy nodes. Bold solid edges are part of expansion trees, dummy-nodes are denoted by squares.

Before we discuss the details, we give an overview of the planarization approach for RMCM.

(1) Compute a planar subgraph $G' = (V, E_P)$ of G.
(2) For each edge $e = \{s, t\} \in E \setminus E_P$:
  (a) Compute S and T.
  (b) Find an insertion path p from S to T in G'.
  (c) Insert e into G' according to p by splitting nodes if required and introducing new dummy nodes for crossings.

It remains to show how to generalize the edge insertion algorithms. In the fixed embedding scenario, we simply introduce a super start node s^* connected to all nodes in S, and a super end node t^* connected from all nodes in T in the search network. The following lemma shows the key property for generalizing the variable embedding case.

Proposition 2. *1. The blocks of G' containing a node in S (T) and the cut vertices of G' contained in S (T) form a subtree of the BC-tree of G'.*
2. Let $\mathcal{T}$ be the SPQR-tree of a block of G'. Then, the nodes of $\mathcal{T}$ whose skeletons contain a node in S (T) form a subtree of $\mathcal{T}$.

This allows us to compute the shortest paths in the BC- and SPQR-trees in a similar way as described above. The only difference is that we consider blocks and skeletons containing any node in S (or T). The computation of insertion paths in R-node skeletons is generalized as for the fixed case if several nodes of S or T are contained.

In [7], two important improvement techniques for the planarization approach are described which are both also applicable for RMCR. The *permutation* strategy calls step (2) several times and processes the edges in $E \setminus E_P$ in random order. The *postprocessing* strategy successively removes an edge path and tries to find a better insertion path. This can also be done for tree paths which in fact is a key optimization of our approach, since it allows to introduce crossings between two tree expansions as well. Finally, we remark that we also contract tree paths during the algorithm if they no longer contain a dummy node and thus become redundant.

5 Optimal Node Insertion

In this section we prove the theorems regarding the minor node insertions.

Algorithm for Theorem 2. We show that MNIF$^-$ is solvable in $\mathcal{O}(|V| \cdot |E'|)$.

Let U be the nodes of V incident to edges of E'. We can solve the node insertion problem with fixed topology for the traditional crossing number by considering the dual graph D of G with respect to Γ. Each node in D is labeled with a number which is initially 0. We then start a BFS for each $u \in U$, augmenting D with edges between u and its incident faces. The nodes of U are incremented by the BFS-depth, for each different u. Finally, each node of D holds the sum of the shortest distances between itself and the nodes U. We then simply pick a node of D with smallest number, and insert the new node v into the corresponding face in Γ.

Using the ideas from the above sections, we can use the same algorithm but allowing edges to cross through nodes. Since all inserted edges are incident to v, they will not cross each other in any optimal node insertion. Therefore, no conflicting edge-node crossings can occur, other then ones based on paths with equal length. Such conflicts can easily be resolved by choosing any of the conflicting paths for both inserted edges. The correctness and running time of the algorithm follows directly. □

Proof of Theorem 3. We show that MNIF$^+$ and MNIV$^+$ are NP-complete.

We can restrict ourselves to MNIF$^+$. Since a planar 3-connected graph has only a unique planar embedding and its mirror, we naturally have NP-completeness for MNIV$^+$ if MNIF$^+$ is NP-complete. We only briefly sketch the idea of the proof.

We reduce to the planar Steiner tree problem, which is known to be NP-complete [6]. Thereby we are given a planar graph D with integer edge-weights and a subset of its nodes are marked as *terminals*. We ask for a weight-minimum tree T connecting all terminals (and possibly some other nodes). Let G be the dual of D, then finding the Steiner tree in D becomes equivalent to finding a treewise expansion for the node v which we want to insert into G minor-monotonously. Since the integer edge-weights can be bounded to be a polynomial in the graph size, we can replace each edge of weight w by a simple path of w edges, thus only polynomially enlarging the given graph. Hence we have NP-completeness for MNIF$^+$. □

6 An ILP Formulation for RMCM

We briefly sketch how to construct an integer linear program to solve the RMCM, and therefore also the hypergraph crossing minimization problem, to optimality. We base our formulation on the ILP for CM, presented in [4]: its main idea is to have variables $x_{e,f}$ for each pair of edges e and f, which are 1 if the edges cross and 0 otherwise. To circumvent problems with the realizability checking of solutions, the graph G is first modified such that each original edge is replaced by a simple path of multiple edges. The ILP is then based on *Kuratowski constraints*, i.e., for each K_5 and $K_{3,3}$ subdivision contained in (partial planarizations of) G we have an inequality which requires at least one crossing.

We can use this ILP by considering tree expansions of G. We replace each node $v \in W$, with $\deg(v) \geq 4$, by a vertex set V'_v with $|V'_v| = 2\deg(v) - 2$. Each edge originally incident to v is incident to a unique vertex of this set. By augmenting V'_v with edges, we can model any treewise connection of the vertices $\{w \in V'_v \mid \deg(w) = 1\}$. Hence, considering the edges E'_v of the complete graph on the set V'_v, we introduce 0/1-variables y_e for all $e \in E'_v$. If such a variable is 1, the corresponding edge is used for the treewise connection of V'_v.

We now require two main types of constraints: we can generalize the Kuratowski constraints straight-forwardly. Let R be the set of edges with y variables which are contained in some Kuratowski subdivision K. We can subtract the term $\sum_{e \in R}(1 - y_e)$ on the right-hand side of the corresponding $\geq$-constraint, i.e., we only require a crossing on K if all its edges are selected.

Additionally, we have to assure the treewise connection for each V'_v. Therefore we require to select exactly $|V'_v| - 1$ edges of E'_v for each set V'_v, and assure connectivity via traditional cut constraints:

$$\forall v \in W, \deg(v) \geq 4, \forall \emptyset \neq S \subset V'_v : \sum_{u \in S, w \in V'_v \setminus S} y_{\{u,w\}} \geq 1.$$

Note that all these constraints can be added dynamically within a Branch-and-Cut framework using known separation routines. Nonetheless, the resulting ILP seems to be too large even for relatively small graphs, and it is therefore mainly of theoretical interest.

7 Experiments

We implemented our algorithms as part of the open-source *Open Graph Drawing Framework* (OGDF, [13]). We conducted two series of experiments on a Windows PC with a Pentium 4 (3.4 GHz) processor and 2 GB RAM.

The first experiment uses the well-known *Rome* benchmark set [1], which has been used for many studies on the traditional crossing number, e.g., [4,7]. It consists of 11528 real-world graphs with 10–100 nodes. We restricted ourselves to the 8013 non-planar graphs with at least 30 nodes, which have an average density of 1.34. Our main focus was to investigate how the minor crossing number compares to the traditional crossing number in real-world settings. Fig. 4 shows the average crossing numbers and minor crossing numbers per graph size. We can see that the minor crossing minimization leads to roughly 35% less crossings on average. While this diagram shows the results for variable embeddings, the diagram looks nearly identical when considering random fixed embeddings, although the absolute crossing numbers are of course a bit higher. In both cases, for the large graphs, the realizing graphs have about 10% more nodes then the original graphs, and roughly 8% of the graphs' nodes are substituted by expansion trees. All graphs can be solved clearly under a second for the fixed embedding case and under 30 seconds for the variable case. For the latter, the 100-node graphs required 5.5 seconds on average.

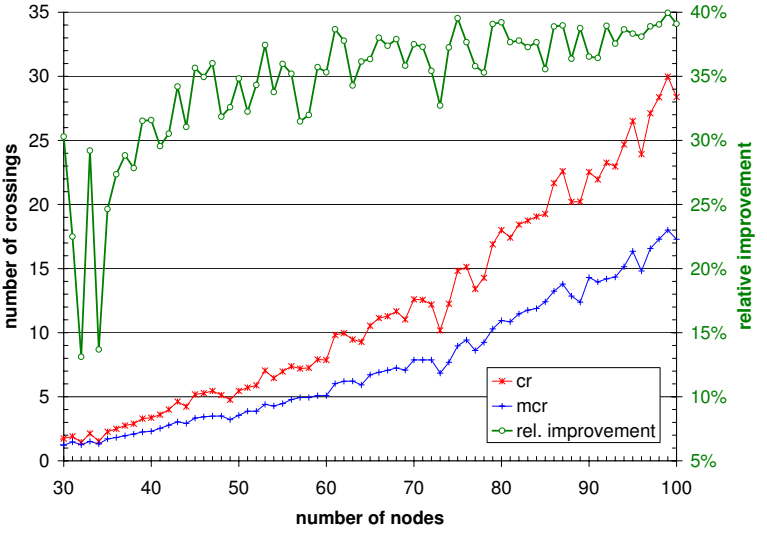


Fig. 4. Results for the Rome graphs (variable embedding)

				FIX				VAR			
name	$\|V\|$	$\|\mathcal{F}\|$	$\|E(H)\|$	phcr	time	thcr	time	phcr	time	thcr	time
b01	43	47	128	32	0.17	18	0.16	33	6.36	18	4.94
b02	25	27	73	12	0.05	8	0.06	12	0.78	8	0.59
b03	148	156	432	148	1.86	58	1.33	131	3:26.91	54	1:32.09
b06	42	50	134	54	0.33	26	0.28	55	13.44	24	7.80
b08	166	179	493	298	8.31	153	4.84	297	18:10.75	146	7:48.16
b09	167	169	472	179	2.27	78	1.77	165	4:47.84	68	2:19.27
b10	183	200	553	431	14.31	212	7.06	432	27:2.30	199	13:42.74
c17	4	11	16	0	0.02	0	0.00	0	0.00	0	0.00
c432	153	196	489	312	6.91	178	4.47	297	15:09.39	167	7:27.97
c499	170	243	578	452	21.19	206	9.78	454	42:55.81	197	20:05.69
s208a	111	122	300	28	0.42	19	0.19	26	15.81	16	9.19
s27a	12	17	33	0	0.00	0	0.00	0	0.00	0	0.00
s298	127	136	385	205	2.97	76	2.14	193	5:42.23	69	2:27.92
s344	164	184	448	57	1.12	38	0.81	57	2:01.20	35	1:01.39
s349	165	185	453	60	0.89	39	0.69	57	1:56.17	37	46.95
s382	173	182	500	206	4.14	88	2.49	186	11:17.64	81	3:58.50
s386a	158	172	511	897	3:22.22	258	16.25	850	147:30.55	238	23:40.38
s400	177	186	518	236	4.53	90	2.55	220	10:06.03	86	3:17.03
s420a	233	252	632	82	2.08	54	1.33	75	4:06.20	47	2:16.05
s444	196	205	569	213	3.77	76	2.11	213	8:31.56	77	3:06.83
s510	210	236	640	1159	2:26.14	489	1:01.41	1096	182:57.00	447	67:21.91
s526a	209	218	675	614	51.83	239	10.39	588	104:08.84	225	27:39.41

The second set of experiments deals with hypergraphs. Therefore we chose all hypergraphs from the ISCA'85, '89, and '99 benchmark sets of real-world electrical networks with up to 500 nodes in their point-based expansion. The following table summarizes our heuristic results for these graphs, considering both phcr, using our traditional crossing minimizer [7], and thcr. The times are given in seconds. We can clearly see the benefit of considering the tree-based drawing style, as compared to the relatively large point-based crossing numbers.

Other papers like [5] considered the tree-based drawing style with certain additional constraints, in particular requiring certain nodes to lie on the "outside" of the drawing. Hence our solutions are not directly comparable. Nonetheless, we think that the numbers show how promising our approach is: the common graphs *s298* and *s400* required 428 and 400 crossings, respectively, using the best algorithm in [5], while our best solutions are 69 and 86, respectively. Hence it seems worthwhile to investigate how to include such constraints into our algorithm.

References

1. Di Battista, G., Garg, A., Liotta, G., Tamassia, R., Tassinari, E., Vargiu, F.: An experimental comparison of four graph drawing algorithms. Comput. Geom. Theory Appl. 7(5-6), 303–325 (1997)
2. Bokal, D., Czabarkab, É., Székely, L.A., Vrt'o, I.: Graph minors and the crossing number of graphs. In: Proc. of 6th Czech-Slovak Int. Symp. on Comb. Graph Theory, Alg. and Appl. ENDM, vol. 28, pp. 169–175 (2007)
3. Bokal, D., Fijavz, G., Mohar, B.: The minor crossing number. SIAM J. Discrete Math. 20, 344–356 (2006)
4. Buchheim, C., Chimani, M., Ebner, D., Gutwenger, C., Jünger, M., Klau, G.W., Mutzel, P., Weiskircher, R.: A branch-and-cut approach to the crossing number problem. In: Healy, P., Nikolov, N.S. (eds.) GD 2005. LNCS, vol. 3843, pp. 37–48. Springer, Heidelberg (2006)
5. Eschbach, T., Günther, W., Becker, B.: Orthogonal hypergraph drawing for improved visibility. J. Graph Algorithms Appl. 10(2), 141–157 (2006)
6. Garey, M.R., Johnson, D.S.: The rectilinear steiner tree problem is NP-complete. SIAM J. Appl. Math. 32, 826–834 (1977)
7. Gutwenger, C., Mutzel, P.: An experimental study of crossing minimization heuristics. In: Liotta, G. (ed.) GD 2003. LNCS, vol. 2912, pp. 13–24. Springer, Heidelberg (2004)
8. Gutwenger, C., Mutzel, P., Weiskircher, R.: Inserting an edge into a planar graph. Algorithmica 41(4), 289–308 (2005)
9. Hliněný, P.: Crossing number is hard for cubic graphs. J. Combin. Theory Ser. B 96, 455–471 (2006)
10. Johnson, D.S., Pollak, H.O.: Hypergraph planarity and the complexity of drawing venn diagrams. J. Graph Theory 11(3), 309–325 (1987)
11. Klemetti, H., Lapinleimu, I., Mäkinen, E., Sieranta, M.: A programming project: Trimming the spring algorithm for drawing hypergraphs. SIGCSE Bull. 27(3), 34–38 (1995)
12. Mäkinen, E.: How to draw a hypergraph. Int. J. Comput. Math. 34, 177–185 (1990)
13. OGDF – Open Graph Drawing Framework (2007), See `http://ls11-www.cs.uni-dortmund.de/ogdf`
14. Sander, G.: Layout of directed hypergraphs with orthogonal hyperedges. In: Liotta, G. (ed.) GD 2003. LNCS, vol. 2912, pp. 381–386. Springer, Heidelberg (2004)
15. Vrt'o, I.: Crossing numbers of graphs: A bibliography (2007), See `ftp://ftp.ifi.savba.sk/pub/imrich/crobib.pdf`

On Mixing and Edge Expansion Properties in Randomized Broadcasting*

Thomas Sauerwald

Department of Computer Science
Fürstenallee 11
33102 Paderborn, Germany
sauerwal@upb.de

Abstract. A very simple and natural broadcasting algorithm is the so-called push algorithm which has several applications in the area of distributed computing. Initially, only one vertex of a graph $G = (V, E)$ owns a piece of information which is spread iteratively to all other vertices: in each time step $t = 1, 2, \ldots$ every *informed* vertex chooses some neighbor uniformly at random which then becomes informed and may itself inform other vertices in the succeeding time steps. The crucial question is how many time steps are required such that all vertices become informed (with high probability).

For various graph classes, involved methods have been developed in order to show an upper bound of $\mathcal{O}(\log N + \mathrm{diam}(G))$, where N is the number of vertices and $\mathrm{diam}(G)$ denotes the diameter of G. However, currently no asymptotically tight bound on the runtime of the push algorithm based on the mixing time exists. In this work we fill this gap by deriving an upper bound of $\mathcal{O}(\log N + \mathsf{T}_{\mathrm{mix}})$, where $\mathsf{T}_{\mathrm{mix}}$ denotes the mixing time of a certain random walk on G. After that we give a simple but useful upper bound which is based on a certain average value of the edge expansion of G. Unfortunately, both approaches do not give the right bound for Hypercubes. Therefore, we develop a general way to combine them and prove that the runtime of the push algorithm is $\Theta(\log N)$ on every Hamming graph.

1 Introduction

Models and Motivation: The study of information spreading in large networks has various fields of application in distributed computing. Consider for example the maintenance of replicated databases on name servers in a large network [3,10]. There are updates injected at various vertices, and these updates must be propagated to all the vertices in the network. In each step, a processor and its neighbors check whether their copies of the database agree, and if not,

* This work was partially supported by German Science Foundation (DFG) Research Training Group GK-693 of the Paderborn Institute for Scientific Computation (PaSCo) and by the Integrated Project IST-15964 "Algorithmic Principles for Building Efficient Overlay Networks" (AEOLUS) of the European Union.

T. Tokuyama (Ed.): ISAAC 2007, LNCS 4835, pp. 196–207, 2007.

they perform the necessary updates. In order to be able to let all copies of the database converge to the same content, efficient broadcasting algorithms have to be developed.

Broadcasting is closely related to certain mathematical models of epidemic diseases where each infected person infects some neighbor chosen uniformly at random [21]. In most of these papers, spreaders are only active in a certain time window, and the question of interest is, whether on certain networks modeling personal contacts an epidemic outbreak occurs. Several threshold theorems involving the basic reproduction number, contact number, and the replacement number have been stated. See e.g. [12] for a collection of results concerning the mathematics of infectious diseases. In contrast to the study of broadcasting, the underlying networks are usually complete graphs [10].

There is an enormous amount of experimental and theoretical study of broadcasting algorithms in various models and on different networks. Several (deterministic and randomized) algorithms have been developed and analyzed. Here, we focus on the time efficiency of randomized broadcasting and study the runtime of the *push algorithm* [3] (and very similar algorithms) defined as follows. Place at the initial time step $t = 0$ a piece of information r on one of the vertices of a graph $G = (V, E)$, where $N := |V|$. Then, in each succeeding time step $t = 1, 2, \ldots$ any *informed* vertex forwards a copy of r to a communication partner over an incident edge selected independently and uniformly at random. The advantage of randomized broadcasting is in its inherent robustness against several kinds of failures and dynamical changes compared to deterministic schemes that either need substantially more time [11] or can tolerate only a relatively small number of faults [15].

Related Work: Most papers dealing with randomized broadcasting analyze the runtime of the push algorithm on different graph classes. Pittel [20] proved that with a certain probability an information is spread to all vertices by the push algorithm within $\log_2 N + \ln N + O(1)$ steps in a complete graph K_N. Feige et al. [10] gave general upper bounds holding for any graph, which were partially improved and complemented by lower bounds in [7]. Moreover Feige et al. determined the runtime on random graphs and Hypercubes up to constant factors. Notably, for Hypercubes the analysis was quite involved. In [8] we analyzed the runtime of the push algorithm on several other Cayley graphs.

We should also note that various broadcasting models have been analyzed in some scenarios that allow vertices and/or edges to fail during the algorithm is executed (e.g. [13]). Most of these papers deal with the worst case asymptotic behavior of broadcasting algorithms when the failures are governed by an adversary, however, in some papers [7] the random failure scenario is also considered.

Due to the simple and natural definition of the push algorithm, we think that this algorithm is of independent interest in theoretical computer science, beyond the applications mentioned earlier. However, the push algorithm is much less explored than random walks on graph. Therefore, we recently tried to reduce the runtime of the push algorithm to the so-called mixing time [18] which measures how fast a random walk converges towards the equilibrium distribution. More

precisely, we showed in [8] that the runtime of the push algorithm is upper bounded by the mixing time of a corresponding random walk and an additional logarithmic factor. Also recently, Boyd et al. [1] considered so-called averaging algorithms and related them to the mixing time of random walks and to a similar broadcasting model.

While the focus of the papers cited above and of this one is only on the time needed to broadcast some information to all nodes of a network, one might also want to minimize the number of messages needed to do so. It was observed in complete graphs of size N that the push algorithm needs at least $\Omega(N \log N)$ transmissions to inform all nodes of the graph, w.h.p. However, in the case of the pull algorithm (in this algorithm uninformed vertices call a neighbor u.a.r., and if this neighbor is informed the node itself becomes informed), if a constant fraction of the nodes are informed, then within $O(\log \log N)$ additional steps every node of this graph becomes informed as well, w.h.p. [3,14]. This implies that in such graphs at most $O(N \log \log N)$ transmissions are needed if the distribution of the information is stopped at the right time. Using this fact, in [14] Karp et al. consider a push and pull algorithm, and presented a termination mechanism in order to bound the number of total transmissions by $O(N \log \log N)$ in complete graphs.

Their analysis has been recently extended to random graphs. In the random graph model considered there, every edge between two vertices exists independently with probability p. Note that these graphs are frequently used for modeling peer to peer networks, e.g. [2]. In [6], it was shown that the number of transmissions is $\Theta(N \log N / \log(pN))$ in case of the push and pull algorithm used by Karp et al. In a very recent work [9], we considered a slightly different algorithm, where each vertex may choose 4 *different* neighbors for push and pull transmission. Surprisingly, this minor change in the ability of the vertices leads to an exponential decrease in the number of transmissions which reduces to $\Theta(N \log \log N)$.

Our Results: The next section contains the basic notation and definitions required for our analysis. In Section 3 we show that the runtime of the push algorithm is asymptotically upper bounded by the sum of the mixing time of a certain random walk on G and $\log N$. In Section 4 we state upper bounds using edge expansion properties of G. Finally, in Section 5 we develop a completely new technique which combines lower bounds on some edge expansion measure with a weaker notion of mixing time. Our method easily applies to Hypercubes and certain generalizations of them. The last section contains our conclusions. Due to space limitations, some proofs are omitted.

2 Notation and Definitions

Let $G = (V(G), E(G))$ denote an unweighted, undirected, simple and connected graph, where $N := |V|$ denotes the size of the graph[1]. In most cases, we will

[1] We prefer to preserve n for the dimension of certain graphs like Hypercubes.

consider families of graphs $G_n = (V_n, E_n)$, where $|V_n| \to \infty$ for $n \to \infty$. By $\mathrm{diam}(G)$ we denote the diameter of G and $N(v)$, $\deg(v)$ denotes the neighborhood and degree of some vertex $v \in V(G)$, respectively. Furthermore, let δ be the minimum and Δ be the maximum degree. By K_i we denote the complete graph formed by i vertices and $\times$ denotes the Cartesian product of two graphs.

As already mentioned before, in this paper we mainly consider the following (parallel) randomized broadcasting algorithm (known as the push algorithm [3]): Place at time $t = 0$ an information r on one vertex s of the graph G. In the succeeding time steps (or rounds) each *informed* vertex forwards a copy of r to a communication partner over an incident edge selected independently and u.a.r. (shorthand for uniformly at random). Throughout this paper, we denote by $I(t)$ the set of informed nodes at time t. With this notation at hand, a formal description of the push algorithm can be found in Figure 1 below. This algorithm will be shortly denoted by $\mathsf{RBA}_{\mathrm{push}}$. Note, that the so-called *pull-algorithm*, $\mathsf{RBA}_{\mathrm{pull}}$, is defined similarly with the only difference that the roles of u and v are interchanged. Here, in each time step each uninformed vertex calls some neighbor u.a.r. and becomes informed if this neighbor has been already informed.

Parallel Push Algorithm	Sequential Push Algorithm
$t \leftarrow 0$	$t \leftarrow 0$
$I(t) \leftarrow \{s\}$	$I(t) \leftarrow \{s\}$
while $I(t) \neq V$ **do**	**while** $I(t) \neq V$ **do**
$\quad I(t+1) \leftarrow I(t)$	$\quad$ choose $u \in V$ u.a.r.
$\quad$ **for** all nodes $u \in I(t)$ **do**	$\quad\quad$ choose $v \in N(u)$ u.a.r.
$\quad\quad$ choose $v \in N(u)$ u.a.r.	$\quad$ **if** $u \in I(t)$ **then**
$\quad\quad I(t+1) \leftarrow I(t+1) \cup \{v\}$	$\quad\quad I(t + \frac{1}{N}) \leftarrow I(t) \cup \{v\}$
$\quad$ **end for**	$\quad$ **end if**
$\quad t \leftarrow t+1$	$\quad t \leftarrow t + \frac{1}{N}$
end while	**end while**

Fig. 1. Definition of the parallel (original) and sequential version of $\mathsf{RBA}_{\mathrm{push}}$. Neglecting constant factors, they have the same runtime on any graph.

Our objective is to determine how many time steps are required to inform every node of G, whereas we make no assumption on the choice of $s \in V$. Let $\mathsf{T}_{\mathrm{push}}(G,p) := \min\{t \in \mathbb{N} \mid \mathbf{Pr}\,[I(t) = V\,] \geq 1-p\}$ denote the runtime of $\mathsf{RBA}_{\mathrm{push}}$ in G, i.e. the number of time steps needed by the push algorithm to inform all vertices of G with probability $1-p$. Note that the push algorithm requires clearly at least $\max\{\log_2 N, \mathrm{diam}(G)\}$ rounds on any graph [10].

All algorithms introduced yet work synchronously and in a parallel fashion. However, it is useful to consider the following sequential push algorithm depicted in Figure 1 whose runtime equals the one of its parallel counterpart, up to a constant factor [8]. Note that in order to make both algorithms comparable, the time axis consists now of discrete sub-time steps $\mathbb{T} := \{i + \frac{j}{N} \mid i \in \mathbb{N}, j \in \{0, \ldots, N-1\}\}$.

We may also consider a sequential broadcasting algorithm $\mathsf{RBA}_{\text{push \& pull}}$ (see also [14] for a similar parallel model) which combines the push and pull transmissions as follows. In each sub-time step $t \in \mathbb{T}$ a vertex $u \in V(G)$ is chosen u.a.r. which then chooses a neighbor $v \in N(u)$ u.a.r. If any of the two vertices is already informed, then both vertices will be informed after this sub-time step.

3 Mixing Time and Broadcasting Time

In this section we relate the mixing time of a certain random walk on G to the runtime of the push-algorithm (For an introduction to random walks and Markov chains, the reader is referred to e.g. [17], [18] or [19]). Typically, a random walk moves in each step to some neighbor selected uniformly at random. More generally, a random walk can described by a stochastic transition matrix $\mathbf{Q}$, with the property that $\mathbf{Q}_{uv} > 0$ only if $\{u, v\} \in E$. Then, a random walk located at some vertex u moves to some adjacent vertex v with probability $\mathbf{Q}_{uv}$. As usual, for any $k \in \mathbb{N}$, $\mathbf{Q}^k$ denotes the k-step transition matrix. If the random walk (or the corresponding Markov chain) is ergodic, i.e. irreducible and aperiodic, it is well-known that the distribution of the random walk converges to its unique stationary distribution vector denoted by $\pi = (\pi_1, \ldots, \pi_N)$. For two given probability vectors $(\mu_i)_{i=1}^N$ and $(\nu_i)_{i=1}^N$ let $\|\mu - \nu\| = \frac{1}{2}\sum_{i=1}^N |\mu_i - \nu_i|$ be the variation distance of these vectors. Then the mixing time captures the speed of convergence, i.e.

$$\mathsf{T}^{\mathbf{Q}}_{\text{mix}}(G, \epsilon) := \min\{t \in \mathbb{N} \mid \|z\mathbf{Q}^t - \pi\| \leq \epsilon \text{ for any probability vector } z\}.$$

We also require the following generalization of the push-algorithm. Recall that in the original push algorithm defined in Section 2, each vertex selects one neighboring vertex u.a.r. Therefore, this selection can be modeled by a matrix $\mathbf{P}$, where

$$p_{uv} = \begin{cases} 0 & \text{if } u = v, \\ \frac{1}{\deg(u)} & \text{if } \{u, v\} \in E, \\ 0 & \text{if } \{u, v\} \notin E. \end{cases}$$

In fact, all stochastic $N \times N$-matrices $\mathbf{Q}$ such that $q_{uv} = 0$ for $\{u, v\} \notin E$ define a variant of the sequential push algorithm $\mathsf{RBA}^{\mathbf{Q}}_{\text{push}}$. Here, for each $t \in \mathbb{T}$ one first chooses one vertex $u \in V$ u.a.r. and then this vertex sends the information to some neighbor chosen randomly according to the u-th row of $\mathbf{Q}$. Notice that $\mathbf{P}$ is perhaps the most natural choice, because all neighbors are selected equiprobable.

In [1] the authors provide a detailed analysis of the convergence rate of certain averaging algorithms by mainly using spectral analysis and methods from convex optimization. By combining Theorem 7 and Theorem 11 in [1] we obtain the following.

Corollary 1. *For any graph G and a symmetric stochastic matrix $\mathbf{Q}$ defining an ergodic random walk it holds*

$$\mathsf{T}^{\mathbf{Q}}_{\text{push\&pull}}(G, N^{-1}) = \mathcal{O}\Big(\log N + \mathsf{T}^{\mathbf{Q}}_{\text{mix}}(G, N^{-2})\Big).$$

Notice that on non-regular graphs, the matrix $\mathbf{P}$ as defined before is *not* symmetric. Therefore, our objectives are to generalize this bound to the non-regular case and to relate the push-algorithm to the push&pull-algorithm. In the following two lemmas we only consider sequential algorithms.

Lemma 1. *If* $\mathbf{Q}$ *is a symmetric and stochastic* $N \times N$*-matrix, then*

$$\mathsf{T}^{\mathbf{Q}}_{\text{push}}(G, N^{-1}) = \mathsf{T}^{\mathbf{Q}}_{\text{pull}}(G, N^{-1}),$$
$$\mathsf{T}^{\mathbf{Q}}_{\text{push}}(G, N^{-1}) = \Theta\Big(\mathsf{T}^{\mathbf{Q}}_{\text{push\&pull}}(G, N^{-1}) + \log N\Big).$$

Lemma 2. *For* $\tilde{\mathbf{P}} := \mathbf{I} - \frac{1}{\Delta} \cdot \mathbf{L}$*, where* $\mathbf{I}$ *is the identity and* $\mathbf{L}$ *is the Laplacian Matrix of* G*, we have*

$$\mathsf{T}^{\mathbf{P}}_{\text{push}}(G, N^{-1}) \leq \mathsf{T}^{\tilde{\mathbf{P}}}_{\text{push}}(G, N^{-1}) = \frac{\Delta}{\delta}\mathcal{O}\Big(\mathsf{T}^{\mathbf{P}}_{\text{push}}(G, N^{-1})\Big).$$

Combining both lemmas and Corollary 1 we arrive at the main result of this section.

Theorem 1. *For any graph* $G = (V, E)$ *we have*

$$\mathsf{T}^{\mathbf{P}}_{\text{push}}(G, N^{-1}) = \mathcal{O}\Big(\log N + \mathsf{T}^{\tilde{\mathbf{P}}}_{\text{mix}}(G, N^{-2})\Big).$$

Lemma 2 says the less $\frac{\Delta}{\delta}$ is, the less we lose in the bound in Theorem 1 by the replacement of $\mathbf{P}$ by $\tilde{\mathbf{P}}$. As shown by the star $K_{1,N-1}$, we have to replace $\mathbf{P}$ by $\tilde{\mathbf{P}}$, because $\mathsf{T}^{\mathbf{P}}_{\text{mix}}(K_{1,N-1}, N^{-1}) = \Theta(\log N)$, but $\mathsf{T}^{\mathbf{P}}_{\text{push}}(K_{1,N-1}, N^{-1}) = \Theta(N \log N)$ due to the coupon collector's problem.

We should mention here, that Theorem 1 gives an optimal bound of $\mathcal{O}(\log N)$, whenever $\mathsf{T}^{\tilde{\mathbf{P}}}_{\text{mix}}(G, N^{-2}) = \mathcal{O}(\log N)$. This is in fact the case for various important graph classes such as complete graphs, expanders [19], and random graphs. Also to several Cayley graphs which occur in the analysis of card shuffling [4], the theorem can be applied.

4 Edge Expansion Properties

In this section we consider the relationship between edge expansion properties and the runtime of the push algorithm. First we define a certain measure which captures the edge expansion properties for subsets of vertices having size m.

Definition 1. *For any graph* G *and any integer* $m \in \{1, \ldots, N-1\}$ *define*

$$\Phi(m) := \min_{X \subseteq V(G), |X| = m} \frac{\sum_{v \in X} \frac{\deg_{X^c}(v)}{\deg(v)}}{N},$$

where $\deg_{X^c}(v)$ *denotes the number of neighbors of* v *within* X^c*. If* G *is regular, then the first formula can be rewritten as*

$$\Phi(m) = \min_{X \subseteq V(G), |X| = m} \frac{|E(X, X^c)|}{2 \cdot |E|}.$$

Here, $E(X, X^c)$ *denotes the set of edges connecting* X *and its complement* X^c*.*

Lemma 3. *Let $X_1, \dots, X_N$ be geometrically distributed random variables with parameters $p_1, \dots, p_N > 0$. Let $X := \sum_{k=1}^{N} X_k$, $\mu := \mathbf{E}[X] = \sum_{k=1}^{N} 1/p_k$ and $p_{min} := \min_{k=1}^{N} p_k$ such that $\mu p_{min} = \Omega(\log N)$. Then $\mathbf{Pr}[X \geq \alpha\mu] \leq N^{-1}$, for some proper constant α.*

Intuitively, it is clear that good edge expansion properties should imply fast broadcasting which is formalized below.

Theorem 2. *For any graph $G = (V, E)$ we have in the sequential model that*

$$\mathbf{E}[\mathsf{T}_{\mathrm{push}}(G)] \leq \sum_{m=1}^{N-1} \frac{1}{N\Phi(m)},$$

where $\mathsf{T}_{\mathrm{push}}(G) := \min\{t \in \mathbb{T} \mid I(t) = V\}$. If for each $1 \leq m' \leq N-1$, $(\sum_{m=1}^{N-1} 1/\Phi(m)) \cdot \Phi(m) = \Omega(\log N)$ then

$$\mathsf{T}_{\mathrm{push}}(G, N^{-1}) = \mathcal{O}\left(\sum_{m=1}^{N-1} \frac{1}{N\Phi(m)}\right).$$

Proof. Let X_i be the waiting time (in sub-time steps) until $|I(t)|$ increased from i to $i+1$. Note, that this time is bounded in the sequential push algorithm by the probability of first choosing a vertex $u \in I(t)$ and then choosing a vertex $v \in N(v) \cap I(t)^c$. Therefore, the probability that $I(t)$ increases by 1 in one sub-time step equals

$$\sum_{v \in I(t)} \mathbf{Pr}[v \text{ chosen}] \cdot \mathbf{Pr}[u \in N(v) \cap I(t)^c \text{ chosen}] = \sum_{v \in I(t)} \frac{1}{N} \frac{\deg_{I(t)^c}(v)}{\deg(v)}.$$

Hence $\mathbf{E}[X_i] \leq 1/\Phi(i)$ and simply summing up over all i and translating this into time steps yields the first claim. The second statement follows directly from Lemma 3. □

The next Proposition shows that the bound of Theorem 2 is almost tight.

Proposition 1. *In the sequential model we have*

1. *If $G = K_{N/2} \times K_2$, where 2 divides N, then $\sum_{m=1}^{N-1} \frac{1}{N\Phi(m)} \leq 2 \ln N$,*
2. *If $G = K_N$, then $\sum_{m=1}^{N-1} \frac{1}{N\Phi(m)} \leq 2\frac{N-1}{N} \ln N$.*
3. *On any graph $G = (V, E)$ we have $\mathsf{T}_{\mathrm{push}}(G, f(N)) \geq \log_{e^{1-\frac{1}{\Delta}}} N + \ln N - o(\ln N) \geq 2 \ln N - o(\ln N)$, where $f(N)$ is a proper function such that $\lim_{N\to\infty} f(N) = 1$.*

Note that we have $\mathsf{T}^{\tilde{\mathbf{P}}}_{\mathrm{mix}}(K_{N/2} \times K_2, \Theta(1)) = \Omega(N)$, but $\mathsf{T}^{\tilde{\mathbf{P}}}_{\mathrm{push}}(K_{N/2} \times K_2, N^{-1}) = \mathcal{O}(\log N)$. This implies that the bound of Theorem 1 overestimates the runtime on this graph drastically. The second point and third point of this proposition imply that complete graphs have (up to low order terms) the lowest broadcasting time among all graphs in the sequential model. Let us compare this result to the parallel model. Here, the runtime on the complete graph is $\log_2 N + \ln N$ by a result of Pittel [20], whereas a tight lower bound of $\log_2 N + \ln N - o(\log N)$ could only be shown for regular graphs [7].

5 Combining Mixing Time and Edge Expansion Properties

Actually, our investigations are motivated by the fact that the bound of Corollary 1 gives not an upper bound of $\mathcal{O}(\log N)$ for the Hypercube, since in this case $\mathsf{T}^{\tilde{\mathbf{P}}}_{\mathrm{mix}}(G, \Theta(1)) = \Theta(\log N \log\log N)$ (cf.[5]). In the same way, the bound based on the edge expansion of Theorem 2 would also give a too weak bound. However, by combining a certain alteration of mixing time with proper lower bounds on $\Phi(m)$ for small m we will overcome these difficulties.

First we require the following technical definition.

Definition 2. *[8] A node $u \in V$ contacts another node $v \in V$ within the time-interval $[a, b]$, if there exists a path $(u_1 := u,\ u_2, \ldots,\ u_{m-1},\ u_m := v)$ with the property $\exists t_1 < t_2 < \cdots < t_{m-1} \in [a, b] : u_i$ chooses u_{i+1} at time $t_i,\ i \in \{1, \ldots, m-1\}$.*

Let us remark that if u is already informed at time a, then v will be also informed at time b.

Before going into further details let us briefly describe our main idea. Suppose that it is easy to show that x (x being some proper polynomial number in N) vertices can be informed in time $\mathcal{O}(\log N)$, e.g. by lower bounds on $\Phi(m)$. Fix some vertex w. By some symmetry considerations, there also have to be x uninformed vertices at some time $t = \mathcal{O}(\log N)$ in order to keep w uninformed at some time $t' = \mathcal{O}(\log N) > t$. Now assign to each informed vertex one random walker and let each random walk follow the directions indicated by the transmissions of the push algorithm. If after $\mathcal{O}(\log N)$ steps, each random walk has a not too small probability to get to an arbitrary vertex, then with a reasonable chance one random walk will be located on one of these at least x uninformed vertices and consequently, w will be informed at time t'.

We will now introduce a weaker notion of the mixing rate which turns out to be very useful. Again it is convenient to consider transition matrices $\mathbf{Q} := \frac{1}{c}\mathbf{I} + (1 - \frac{1}{c})\mathbf{P}$, where $c' > 0$, to avoid periodicity problems.

Definition 3. *For any graph G let $\Psi^{\mathbf{Q}}_t(G) := \min_{u,v \in V} \mathbf{Q}^t_{uv}$.*

Observe that for every graph $\Psi^{\mathbf{Q}}_{T^{\mathbf{Q}}_{\mathrm{mix}}(G,N^{-2})}(G) \geq N^{-1} - N^{-2}$. However, Theorem 3 below only requires $\Psi^{\mathbf{Q}}_t(G) \geq N^{-\beta}$, where β is some proper constant between 1 and 2.

Theorem 3. *Suppose that a family of regular graphs satisfies the following properties,*

1. *N^{α} vertices can be informed in time t_1 w.p. $1 - N^{-1}$, regardless of the initial placement of the information,*
2. *$\Psi^{\mathbf{Q}}_{t_2} \geq N^{-\beta}$, where $t_2 = \mathcal{O}(\mathrm{polylog}\, N)$,*
3. *$0 < \alpha < 1$ and $\beta < 2$ are constants such that $\beta < 3\alpha - 1$.*

Then we have that

$$\mathsf{T}^{\mathbf{P}}_{\mathrm{push}}(G, N^{-1}) = \mathcal{O}(t_1 + t_2).$$

Proof. First note that by Lemma 1 we may replace the matrix $\mathbf{P}$ by the matrix $\mathbf{Q}$ which only makes the push algorithm slower. Using the first condition, we have that $|I(t_1)| \geq N^{\alpha}$ w.p. $1 - N^{-1}$. Then let $t' = 2t_1 + \gamma t_2$, where $\gamma > 0$ is a sufficiently large constant. Fix some arbitrary vertex w. For some $t \leq t'$ let $C_w(t)$ be the set of vertices which contact w within the time-interval $[t, t']$. Since G is regular and $\mathbf{Q}$ is symmetric, the sequence $C_w(t)$ with decreasing t evolves in the same way as $I(t)$ with increasing t. Therefore, for a large enough γ, $|C_w(t' - t_1)| \geq N^{\alpha}$ holds with probability $1 - N^{-1}$. Let $\mathcal{A}$ be the event that $|I(t_1)| \geq N^{\alpha}$ and $|C_w(t' - t_1)| \geq N^{\alpha}$ occur.

Consider now the following model of interacting random walks (see also [8]). There are m concurrent random walks on the graph. At each sub-time step $t \in \mathbb{T}$, some vertex is chosen u.a.r., and if it hosts one or more random walks, then exactly one random walk is allowed to perform a transition according to $\mathbf{Q}$. Here, ties can be broken by the FIFO-rule, for example. In [8] we already remarked that the sequential push algorithm can obviously simulate these m interacting random walks. In particular, the random walks spread the information to the vertices of G, if they are initially located at some informed vertex. Now distribute at time t_1 N^{α} random walks among V, where each initial position is chosen independently and u.a.r. Denote by R the set of the initial positions of these m random walks. Let $\mathcal{B}$ be the event, that $|I(t_1) \cap R| \geq \frac{3}{4} N^{2\alpha - 1}$ occurs. By a Chernoff-Bound [18, p.66], we get

$$\mathbf{Pr}\left[\, |I(t_1 \cap R)| \leq \frac{3}{4} N^{2\alpha-1} \,\right] \leq e^{-\frac{N^{2\alpha-1}}{32}}$$

and therefore $\mathbf{Pr}\,[\,\mathcal{B}\,] \geq 1 - N^{-1}$, if N is large enough.

Then consider only the first t_2 transitions of each of these random walks where we only count sub-time steps in which the corresponding random walk moves according to $\mathbf{Q}$. After some random walk has performed t_2 transitions, we simply remove it from the graph G. Let us consider the probability that one fixed random walk will visit some vertex u during these t_2 transitions:

$$\begin{aligned}\mathbf{Pr}\,[\,u \text{ will be visited}\,] &\leq \sum_{v \in V} \mathbf{Pr}\,[\,\text{Random walk starts at } v\,] \sum_{t=1}^{t_2} \mathbf{Q}^t_{vu} \\ &= \frac{1}{N} \sum_{t=1}^{t_2} \sum_{v \in V} \mathbf{Q}^t_{vu} = \frac{t_2}{N},\end{aligned}$$

since the matrix $\mathbf{Q}$ is doubly stochastic due to the regularity of G. Therefore, the probability that some fixed vertex is visited by at least $\frac{2}{1-\alpha}$ different random walks during their first t_2 transitions is bounded by

$$\binom{N^\alpha}{\frac{2}{1-\alpha}}\left(\frac{t_2}{N}\right)^{\frac{2}{1-\alpha}} \leq N^{\alpha\frac{2}{1-\alpha}}N^{-\frac{2}{1-\alpha}}t_2^{\frac{2}{1-\alpha}}$$
$$\leq N^{\frac{2}{1-\alpha}\cdot(\alpha-1)}t_2^{\frac{2}{1-\alpha}} = N^{-2}t_2^{\frac{2}{1-\alpha}} \leq N^{-\frac{3}{2}},$$

provided that N is large enough. Let $\mathcal{C}$ be the event that no vertex is visited by more than $\frac{2}{1-\alpha}$ different random walks. Hence, by the Union bound [18] over all vertices, we get $\mathbf{Pr}[\,\mathcal{C}\,] \geq 1 - N^{-\frac{1}{2}}$. If $\mathcal{C}$ holds, then all random walks succeed in performing t_2 transitions after time $\gamma := \mathcal{O}(\frac{2}{1-\alpha}t_2 + \log N)$. Let $\mathcal{D}$ denote the event that at least one random walk starting at $I(t_1)$ is located on one of the vertices of $C_w(t'-t_1)$ after having performed t_2 transitions in G. Then

$$\mathbf{Pr}[\,\mathcal{D}\,] \geq 1-\left(1-|C_w(t'-t_1)|\cdot\Psi_{t_2}\right)^{|I(t_1)\cap R|} \geq 1-\left(1-\frac{1}{N^{\beta-\alpha}}\right)^{\frac{1}{4}(N^{2\alpha-1})}$$
$$\geq 1-e^{-N^\epsilon},$$

where $\epsilon > 0$ is a proper constant, since $\beta - \alpha < 2\alpha - 1$. Putting everything together, we have shown that

$$\mathbf{Pr}[\,w \in I(t')\,] \geq \mathbf{Pr}[\,\mathcal{A}\wedge\mathcal{B}\wedge\mathcal{C}\wedge\mathcal{D}\,] \geq 1 - 2N^{-1} - N^{-1} - N^{-1/2} - e^{-N^\epsilon},$$

which implies $\mathbf{Pr}[\,w \in I(5t')\,] \geq 1 - N^{-2}$. As a consequence, the expected number of uninformed vertices after $5t'$ time steps is N^{-1} and an application of Markov's inequality yields the claim, as $t_1 = \Omega(\log n)$. □

Note that it is straightforward, but tedious to reformulate Theorem 3 for non-regular graphs.

We will now introduce the family of so-called Hamming graphs consisting of the n-dimensional Hypercube as a special case for $c = 2$.

Definition 4. *For any two integers $c \geq 2, n \geq 1$ denote by $H(c,n) = K_c^n = \prod_{k=1}^n K_c$ the (c,n)-Hamming graph, where K_i denotes the complete graph formed by i vertices.*

A moment's reflection shows that each $H(c,n)$ has $N = c^n$ vertices, is regular of degree $(c-1)n$ and has a diameter of n. Note also that vertices can be represented as vectors of $[1,c]^n$. Furthermore, we state the following corollary of the isoperimetric numbering of Hamming graphs given by Lindsey [16].

Corollary 2. *For every Hamming graph $H(c,n)$ we have*

$$\Phi(m) \geq \frac{m\cdot(c-1)\cdot\left(n-\lceil\log_c m\rceil\right)}{c^n\cdot(c-1)\cdot n}.$$

It is also not too difficult to bound $\Psi_{6\,\mathrm{diam}} = \Psi_{6n}$ by relating it to the famous coupon collector's problem.

Lemma 4. *For any $H(c,n)$ we have $\Psi_{6\,\mathrm{diam}}(H(c,n)) \leq N^{-\frac{3}{2}}$, whenever N is large enough.*

Corollary 3. *For any $H(c,n)$ it holds that $\mathsf{T}^P_{\text{push}}(H(c,n), N^{-1}) = \mathcal{O}(\log N)$.*

Proof. Using Corollary 2 and the methods in the proof of Theorem 2, the first condition of Theorem 3 holds for *any* constant $\alpha < 1$ and $t_1 = \mathcal{O}(\log N)$. Furthermore, Lemma 4 gives $t_2 = \mathcal{O}(\log N)$ and $\beta = \frac{3}{2} < 2$, so the claim follows by Theorem 3. □

6 Conclusion

In this paper we derived results relating the push algorithm to the mixing time of random walks and to edge expansion properties of graphs. In Section 3 we upper bounded the runtime of this randomized broadcasting algorithm by the sum of the mixing time and $\log N$. In Section 4 we introduced a certain edge expansion measure and bounded the runtime by this value. Since for some graphs the aforementioned methods separately may not result in tight bounds, we combined them in Section 5. Our general finding there easily applies to Hamming graphs and gives an optimal upper bound of $\mathcal{O}(\log N)$.

While in this paper, the push algorithm was related to the mixing time of random walks, it might be also interesting to study its relationship to the cover time (expected number of steps after a random walk has visited all vertices).

References

1. Boyd, S., Ghosh, A., Prabhakar, B., Shah, D.: Randomized Gossip Algorithms. IEEE Transactions on Information Theory and IEEE/ACM Transactions on Networking 52(6), 2508–2530 (2006)
2. Cooper, C., Dyer, M., Greenhill, C.: Sampling regular graphs and peer-to-peer networks. In: 16th ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 980–988 (2005)
3. Demers, A., Greene, D., Hauser, C., Irish, W., Larson, J., Shenker, S., Sturgis, H., Swinehart, D., Terry, D.: Epidemic algorithms for replicated database maintenance. In: 6th ACM Symposium on Principles of Distributed Computing (PODC), pp. 1–12 (1987)
4. Diaconis, P.: Group Representations in Probability and Statistics. In: Lecture notes-Monograph Series, vol. 11 (1988)
5. Diaconis, P., Graham, R.L., Morrison, J.A.: Asymptotic Analysis of a Random Walk on a Hypercube with Many Dimensions. Random Structures and Algorithms 1(1), 51–72 (1990)
6. Elsässer, R.: On the Communication Complexity of Randomized Broadcasting in Random-like Graphs. In: 18th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA), pp. 148–157 (2006)
7. Elsässer, R., Sauerwald, T.: On the Runtime and Robustness of Randomized Broadcasting. In: Asano, T. (ed.) ISAAC 2006. LNCS, vol. 4288, pp. 349–358. Springer, Heidelberg (2006)
8. Elsässer, R., Sauerwald, T.: Broadcasting vs. Mixing and Information Dissemination on Cayley Graphs. In: Thomas, W., Weil, P. (eds.) STACS 2007. LNCS, vol. 4393, pp. 163–174. Springer, Heidelberg (2007)

9. Elsässer, R., Sauerwald, T.: On the Power of Memory in Randomized Broadcasting. In: 19th ACM-SIAM Symposium on Discrete Algorithms (SODA) (to appear, 2008)
10. Feige, U., Peleg, D., Raghavan, P., Upfal, E.: Randomized Broadcast in Networks. Random Structures and Algorithms 1(4), 447–460 (1990)
11. Gąsieniec, L., Pelc, A.: Adaptive broadcasting with faulty nodes. Parallel Computing 22, 903–912 (1996)
12. Hethcore, H.: Mathematics of infectious diseases. SIAM Review 42, 599–653 (2000)
13. Hromkovič, J., Klasing, R., Pelc, A., Ružička, P., Unger, W.: Dissemination of Information in Communication Networks. Springer, Heidelberg (2005)
14. Karp, R., Schindelhauer, C., Shenker, S., Vöcking, B.: Randomized Rumor Spreading. In: 41st IEEE Symposium on Foundations of Computer Science (FOCS), pp. 565–574 (2000)
15. Leighton, T., Maggs, B., Sitamaran, R.: On the fault tolerance of some popular bounded-degree networks. In: 33rd IEEE Symposium on Foundations of Computer Science (FOCS), pp. 542–552 (1992)
16. Lindsey, J.H.: Optimal Assignment of Numbers to Vertices. Amer. Math. Monthly 7, 508–516 (1964)
17. Lovász, L.: Random walks on graphs: A survey. Combinatorics, Paul Erdös is Eighty 2, 1–46 (1993)
18. Mitzenmacher, M., Upfal, E.: Probability and Computing. Cambdrige University Press, Cambdrige (2005)
19. Motwani, R., Raghavan, P.: Randomized Algorithms. Cambridge University Press, Cambridge (1995)
20. Pittel, B.: On spreading a rumor. SIAM Journal on Applied Mathematics 47(1), 213–223 (1987)
21. Kermack, A.M.W.O.: Contributions to the mathematical theory of epidemics. Proc. Roy. Soc., 700–721 (1927)

Linear Reconfiguration of Cube-Style Modular Robots

Greg Aloupis[1], Sébastien Collette[2], Mirela Damian[3], Erik D. Demaine[4], Robin Flatland[5], Stefan Langerman[6], Joseph O'Rourke[7], Suneeta Ramaswami[8], Vera Sacristán[9,*], and Stefanie Wuhrer[10]

[1] Université Libre de Bruxelles, Belgique
greg@scs.carleton.ca
[2] Université Libre de Bruxelles, Belgique
sebastien.collette@ulb.ac.be
[3] Villanova University, Villanova, USA
mirela.damian@villanova.edu
[4] Massachusetts Institute of Technology, Cambridge, USA
edemaine@mit.edu
[5] Siena College, Loudonville, N.Y., USA
flatland@siena.edu
[6] Université Libre de Bruxelles, Belgique
stefan.langerman@ulb.ac.be
[7] Smith College, Northampton, USA
orourke@cs.smith.edu
[8] Rutgers University, Camden, USA
rsuneeta@camden.rutgers.edu
[9] Universitat Politècnica de Catalunya, Barcelona, Spain
vera.sacristan@upc.edu
[10] Carleton University, Ottawa, Canada
swuhrer@scs.carleton.ca

Abstract. In this paper we propose a novel algorithm that, given a source robot S and a target robot T, reconfigures S into T. Both S and T are robots composed of n atoms arranged in $2 \times 2 \times 2$ meta-modules. The reconfiguration involves a total of $O(n)$ atom operations (expand, contract, attach, detach) and is performed in $O(n)$ parallel steps. This improves on previous reconfiguration algorithms [1,2,3], which require $O(n^2)$ parallel steps. Our algorithm is in place; that is, the reconfiguration takes place within the union of the bounding boxes of the source and target robots. We show that the algorithm can also be implemented in a synchronous, distributed fashion.

1 Introduction

A self-reconfiguring modular robot consists of a large number of independent units that can rearrange themselves into a structure best suited for a given environment or task. For example, it may reconfigure itself into a thin, linear

* Partially supported by projects MEC MTM2006-01267 and DURSI 2005SGR00692.

T. Tokuyama (Ed.): ISAAC 2007, LNCS 4835, pp. 208–219, 2007.

shape to facilitate passage through a narrow tunnel, transform into an emergency structure such as a bridge, or surround and manipulate objects in outer space. Since modular robots are comprised of groups of identical units, they can also repair themselves by replacing damaged units with functional ones. Such robots are especially well-suited for working in unknown and remote environments.

Various types of units for modular robots have been designed and prototyped in the robotics community. These units differ in shape and the operations they can perform. In this paper, we consider homogeneous self-reconfiguring modular robots composed of cubical units (*atoms*) arranged in a lattice configuration. Each *atom* is equipped with an expansion/contraction mechanism that allows it to extend its faces out and retract them back. Each face of an atom is equipped with an attaching/detaching mechanism that allows it to attach to (or detach from) the face of an adjacent atom. Prototypes of cubical atoms include crystalline atoms [4] and telecube atoms [5]. The collection of atoms composing a robot is *connected* in the sense that its dual graph (vertices correspond to atoms, edges correspond to attached atoms) is connected. When groups of atoms perform the four basic *atom operations* (expand, contract, attach, detach) in a coordinated way, the atoms move relative to one another, resulting in a reconfiguration of the robot. To ensure connectedness of the reconfiguration space, the atoms are arranged in *meta-modules*, which are groups of $k \times k \times k$ atoms attached to one another in a cubic shape.

The complexity of a reconfiguration algorithm can be measured by the number of *parallel steps* performed, as well as the total number of atom operations. In a parallel step, many atoms may perform moves simultaneously. Reducing the number of parallel steps has a significant impact on the reconfiguration time, because the mechanical actions (expand, contract, attach, detach) performed by the atoms are typically the slowest part of the system. Furthermore, since atoms may have limited battery power, it is useful to reduce the total number of mechanical operations (i.e., the atom operations) performed.

Our main contribution in this paper is a novel algorithm that, given a source robot S and a target robot T, each composed of n atoms arranged in $2 \times 2 \times 2$ meta-modules[1], reconfigures S into T in $O(n)$ parallel steps and a total of $O(n)$ atom operations. Our algorithm improves significantly the previously best-known reconfiguration algorithms for cube-style modular robots [1,2,3], which take $O(n^2)$ parallel steps as well as $O(n^2)$ atom operations. In addition, our algorithm reconfigures S into T in place, in the sense that the reconfiguration takes place within the union of the bounding boxes of S and T, while keeping the robot connected at all times during the reconfiguration. An in place reconfiguration is useful when there are restrictions on the amount of space that a robot may occupy during the reconfiguration process. Note that in this work we have not taken into consideration any issues regarding the robot's mass or inertia. However, the "in place" nature of our algorithms mitigates some of the issues arising from such constraints.

[1] Throughout the paper, n refers to the number of robot atoms and m refers to the number of robot meta-modules, where $n = 8m$.

2 Preliminaries

2.1 Robots as Lattices of Meta-modules

There exist atom configurations which cannot be reconfigured, e.g. a single row of atoms. Connectedness of the reconfiguration space is guaranteed for robots composed of *meta-modules* [1,2], where a meta-module is a connected set of k^3 atoms arranged in a $k \times k \times k$ grid. It is desirable that meta-modules be composed of as few atoms as possible. In our reconfiguration algorithms, meta-modules are of minimum size consisting of a $2 \times 2 \times 2$ grid of atoms [6,2].

We define two basic meta-module moves (hardware independent) used by our reconfiguration algorithms, similar to the ones described in [2].

SLIDE(***dirSlide***)**.** Slides a meta-module one step in the direction *dirSlide* with respect to some substrate meta-modules. This move is illustrated in Fig. 1, where each box represents a meta-module. The preconditions for applying this move are: (i) the sliding meta-module (A in Fig. 1a) is adjacent to a meta-module in a direction orthogonal to *dirSlide* (B in Fig. 1a), which in turn is adjacent to a meta-module in direction *dirSlide* (C in Fig. 1a) and (ii) the target position for the sliding meta-module is free. This move allows the

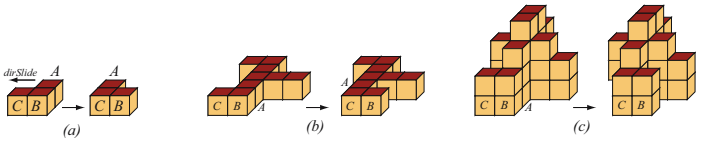


Fig. 1. Examples of SLIDE(x^-): (a) Meta-module A slides alone, (b,c) A carries adjacent meta-modules

sliding meta-module to "carry" other attached meta-modules (as in Figs. 1b-c), as long as the target position for a carried meta-module is unoccupied and the carried meta-module is only attached to other meta-modules moving simultaneously in the same direction.

k-TUNNEL(***sPos***, ***ePos***)**.** Pushes the meta-module located at *sPos* into the robot, and pops a meta-module out of the robot in position *ePos*. There are two preconditions for applying this move: (i) *sPos* is at a leaf node in the dual graph of the starting configuration (i.e. it is attached to only one other meta-module) and *ePos* is a leaf node in the dual graph of the ending configuration, and (ii) there is an orthogonal path through the robot starting at *sPos* and ending at *ePos*, with k orthogonal turns (see Fig. 2). This move performs an "inchworm" move between successive turns. Thus the contracted "mass" of *sPos* is transferred between turns using $O(1)$ motions.

In [7], sequences of atom operations implementing SLIDE and k-TUNNEL for cube-style robots are illustrated. The robot stays connected at all times during a

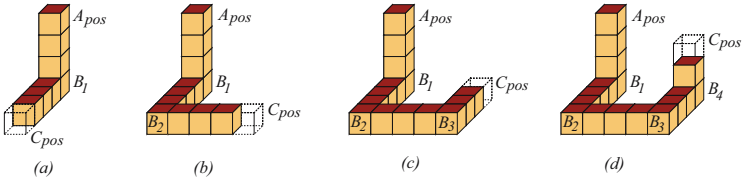


Fig. 2. Examples of TUNNEL(A_{pos}, C_{pos}) with orthogonal turns at B_i, $i = 1, 2, 3, 4$. (a) 1-TUNNEL (b) 2-TUNNEL (c) 3-TUNNEL (d) 4-TUNNEL

meta-module slide or tunnel move. In addition to these two moves, meta-modules can also attach to and detach from adjacent meta-modules.

As for the complexity, attaching and detaching is done in $O(1)$ parallel steps using $O(1)$ atom operations. The SLIDE operation is also implemented in $O(1)$ parallel steps using $O(1)$ atom operations, no matter how many meta-modules are carried in the move. The k-TUNNEL is implemented in $O(k)$ parallel steps using $O(k)$ atom operations, as long as no meta-modules are attached along the path between consecutive turns. Our algorithms ensure this property and only have the need for $k \leq 4$.

2.2 Centralized and Distributed Complexity

We consider both centralized and distributed models of computation. In the centralized model algorithms (described in Sect. 3), computation is performed only by a central processing unit in order to determine the sequence of reconfiguration moves for each meta-module. In Sect. 4 we briefly discuss how to adapt our algorithms to a synchronous distributed model. While this model does not depend on a central processor, it assumes the existence of a clock, used to synchronize the meta-module moves; each meta-module performs local computations to determine the sequence of moves it needs to perform synchronously.

In this paper we do not address the issue of reducing the computation time; however, we observe that straightforward implementations of our centralized algorithms require $O(n^2)$ computation time. The amount of computation performed by each meta-module in the distributed implementations is $O(n)$. Communication time in both models depends on whether information can be broadcasted to all atoms simultaneously, or if information must propagate through the network of atoms. Since a total of $O(n)$ information must be communicated, this takes $O(n)$ time if broadcasted and $O(n^2)$ if propagated.

3 Centralized Reconfiguration

In this section we present an algorithm that reconfigures any given source robot, S, into any given target robot, T, where S and T are each a connected set of m meta-modules composed of $n = 8m$ atoms. We describe the algorithm first for reconfiguring 2D robots which consist of a single layer of meta-modules (Sect. 3.1). We then generalize this to 3D robots (Sect. 3.2).

3.1 Centralized Reconfiguration in 2D

The main idea behind the algorithm is to transform the source robot S into the *common comb* configuration which is defined in terms of both S and T. Then by executing in reverse the meta-module moves of this algorithm for T, we can transform the common comb into T. In transforming S into the common comb, there is an intermediate step in which S is reconfigured into a (regular) *comb*.

2D Robot to 2D Comb. In a comb configuration, the meta-modules form a type of histogram polygon [8]. Specifically, the meta-modules are arranged in adjacent columns, with the bottom meta-module of each column in a common row (see Fig. 3e). This common row is called the *handle*; the columns of meta-modules extending upward from the handle are called *teeth*.

Initially, the algorithm designates the row containing the topmost meta-modules of S as the *wall* (see Fig. 3a). We view the wall as infinite in length. The wall sweeps over the entire robot, moving down one row in each step. By having certain meta-modules slide downward with the wall, the teeth of the comb emerge above the wall. We call this process "combing" the robot. In what follows we will refer to the row of meta-modules immediately above (below) the wall as w^+ (w^-).

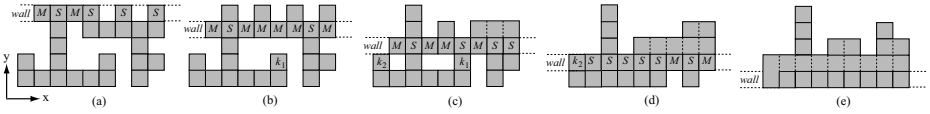


Fig. 3. The initial configuration is converted into a comb as it is swept by the wall

Algorithm 1 outlines the combing process. After initializing the wall in Step 1, the loop in line 2 slides the wall down row by row. In each iteration, Step 2.1 labels each wall meta-module as *stationary* (S) if it has a meta-module adjacent below and *moving* (M) otherwise (see Fig. 3). Intuitively, moving meta-modules will move downward to occupy the gap below. Step 2.2 identifies *moving wall components*, which are maximal sequences of adjacent moving wall meta-modules. In Fig. 3b for example, there are three moving wall components consisting of the 1^{st}, $3^{rd} - 6^{th}$, and 8^{th} wall meta-modules. A moving wall component will always have a stationary meta-module adjacent to one or both ends, for otherwise it would be disconnected from the rest of the robot.

Step 2.3 moves the wall down by one meta-module row. The moving components and the teeth attached to them move down with the wall. This is done by having each moving wall meta-module adjacent to a stationary meta-module perform a SLIDE(y^-) move, thus moving itself one row below w.r.t. the adjacent stationary wall meta-module. Figures 3a-3e show the robot configuration after successive moving wall steps.

A series of attach and detach operations in Step 2.4 prepares the robot for the next iteration. First, the end meta-modules of the moved components attach on

the left and right to any newly adjacent meta-modules (if not already attached). Then each stationary meta-module (now in row w^+) detaches itself from any adjacent meta-modules to its left and right. Finally, all meta-modules in w^- that are now adjacent to a wall meta-module attach to this wall meta-module.

Algorithm 1. 2D-COMBING(S)

1.Set wall to row containing topmost meta-modules of S.
2.**while** there are meta-modules below the wall **do**
  2.1 Label wall meta-modules moving or stationary.
  2.2 Identify moving wall components.
  2.3 Move wall one row lower, carrying moving components and attached teeth.
  2.4 Adjust meta-module attachments

Lemma 1. *The robot configuration forms one connected component at all times.*

Proof. Omitted.

Lemma 2. *A 2D robot can transform into its comb configuration in place in $O(n)$ parallel steps and a total of $O(n)$ atom operations.*

Proof. Clearly the reconfiguration is within the bounding box of the source robot. For each of the $O(m)$ iterations, it performs one parallel set of meta-module SLIDE operations and three parallel attachment operations, which is $O(m) = O(n)$ parallel steps. We now consider the total number of atom operations performed. For each stationary meta-module that emerges above the wall, there are at most 2 moving meta-modules that slid past it, one on either side. At most m stationary meta-modules emerge above the wall, so the total number of SLIDE operations is bounded by $2m$. Since a meta-module is in w^+ and w^- at most once and enters the wall at most once, the number of meta-module attach and detach operations done in Step 2.4 is $O(m)$. The SLIDE and attach/detach operations require $O(1)$ atom operations, making the total number of atom operations performed $O(m) = O(n)$. □

2D Comb to 2D Common Comb. For two combs C_S and C_T, this section describes an algorithm to reconfigure C_S into the *common comb*, an intermediate configuration defined in terms of both C_S and C_T.

Let h_S and h_T be the number of meta-modules in the handles of C_S and C_T, and let $h = \max(h_S, h_T)$. Let $S_1, S_2, \ldots, S_h$ denote the teeth of C_S. If $h_S < h_T$, then let $S_{h_S+1}, \ldots, S_h$ be simply "empty teeth". $|S_i|$ is the number of meta-modules on top of the handle meta-module in tooth S_i; it does not count the handle meta-module. We will represent meta-modules by their "coordinates" in the lattice. When referring to meta-modules by their coordinates, we'll assume the comb's leftmost handle meta-module is at $(1, 1)$. So the set $\{(i, j) \mid 2 \leq j \leq |S_i|+1\}$ is the set of meta-modules in tooth S_i. All terms are defined analogously for comb C_T and for comb C_U, whose description follows.

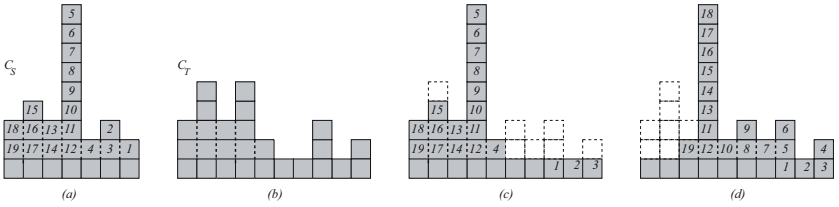


Fig. 4. (a) C_S, with meta-modules labeled in reverse lexicographical order. (b) C_T (c) Shaded meta-modules are C_S after extending its handle's length to match C_U. C_U consists of all shaded and unshaded boxes. Labels indicate which meta-modules moved to form the handle. (d) Shaded meta-modules form the common comb for C_S and C_T.

Let C_U be a comb that is the union of C_S and C_T in the sense that the length of C_U's handle is h and its ith tooth has length $\max(|S_i|, |T_i|)$, $1 \leq i \leq h$. The common comb is a subset of C_U consisting of its h handle meta-modules and a 'right-fill' of the $m - h$ teeth meta modules into the shell defined by C_U. For example, Figs. 4a and 4b show C_S and C_T. In Fig. 4d, C_U consists of all the shaded and unshaded meta-modules; the common comb is all the shaded boxes.

Algorithm 2 describes in detail the process of converting C_S to the common comb. Step 1 initializes queue O with the teeth meta-modules of C_S in reverse lexicographical order on their coordinates. (See the labeled ordering in Fig. 4a.) This is the order in which teeth will be moved to fill in missing meta-modules in the common comb. Step 2 lengthens C_S's handle so that it contains h meta-modules, moving meta-modules from O to the handle using 1-Tunnel operations. Figure 4c shows the results of Step 2.

Once the handle is the proper length, then C_S's teeth are lengthened to match the lengths of C_U's teeth, starting with the rightmost tooth. Since C_U is the union of C_S and C_T, each tooth S_i of C_S is either the same length as the corresponding tooth in C_U, or it is shorter. A key invariant of the algorithm is that at the beginning of an iteration in Step 3, O contains exactly those meta-modules in teeth $S_1, \ldots S_i$ of C_S. This is certainly true in the first iteration when $i = h$, and can be easily shown to be true inductively for all i. Therefore, at the start of an iteration, if $|S_i| > 0$ then the next $|S_i|$ meta-modules in O are exactly the teeth meta-modules in S_i. These meta-modules are already in their final locations, and so they are just removed from O (Loop 3.1). Loop 3.2 then moves the next $|U_i| - |S_i|$ teeth meta-modules in O to tooth S_i using 2-Tunnel operations. Figure 4d shows the resulting common comb.

Observe that in Loop 3.2, tooth *oPos* is always the top meta-module of the first non-empty tooth to the left of tooth S_i. Therefore, the orthogonal path followed in the 2-Tunnel operation is from *oPos* down to the handle meta-module at the base of the tooth, through a (possibly length 0) section of the handle containing only empty teeth, and then up to the top of tooth i. No meta-modules are attached between turns along this path, so the 2-Tunnel operation requires only $O(1)$ basic operations to complete.

Algorithm 2 2D-Comb-To-Common-Comb(C_S, C_U)

1. Let O be a queue of the (i, j) coordinates of the teeth meta-modules (i.e., $j > 1$) of C_S, in reverse lexicographical order.
2. If $h_S < h$ then { extend C_S's handle to length h }
   - 2.1 For $i = h_S + 1$ to h
     - 2.1.1 oPos = $O.dequeue()$
     - 2.1.2 In C_S, 1-Tunnel($oPos$,$(i, 1)$)
3. For $i = h$ down to 1 { lengthen teeth of C_S, from right to left }
   - 3.1 For $j = 1$ to $|S_i|$ $O.dequeue()$ { remove meta-modules already in tooth S_i }
   - 3.2 For $j = |S_i| + 1$ to $|U_i|$ { lengthen tooth S_i }
     - 3.2.1 if $O.size() = 0$ then exit
     - 3.2.2 $oPos = O.dequeue()$
     - 3.2.3 In C_S, 2-Tunnel($oPos$,(i, j))

Lemma 3. *A 2D robot can transform into a common comb configuration in place in $O(n)$ parallel steps and a total of $O(n)$ atom operations.*

Proof. The reconfiguration takes place within the union of the bounding boxes of C_S and C_T, which is contained within the union of the bounding boxes of S and T. At most m modules are relocated, each by a 1-Tunnel or 2-Tunnel operation requiring $O(1)$ atom operations, resulting in $O(m) = O(n)$ parallel steps and atom operations. □

Overall 2D Reconfiguration Algorithm. The general algorithm to reconfigure any m meta-module robot S to any other m meta-module robot T consists of four major steps. First S reconfigures into comb C_S, then C_S reconfigures into common comb C_{ST}. Then the reverse moves of the 2D-Comb-To-Common-Comb and 2D-Combing algorithms reconfigure C_{ST} into C_T and then C_T into T.

Theorem 1. *Any 2D source robot can be reconfigured into any 2D target robot in place in $O(n)$ parallel steps and a total of $O(n)$ atom operations.*

3.2 Centralized Reconfiguration in 3D

Analogous to the 2D case, in 3D the source robot S is also transformed into a 3D common comb and then into target robot T. In transforming to the 3D common comb there are two intermediate configurations, a terrain configuration and a (regular) 3D comb configuration.

Source Robot to 3D Terrain. We use the 3D analog of the 2D-Combing process, 3D-Combing, to reconfigure S into a 3D terrain. The 3D algorithm is the same as in 2D, except the wall now consists of an entire 2D horizontal layer of meta-modules, initially the topmost single layer of S. See Fig. 5. The final result is that all meta-modules of S having the same (x, y) coordinates are grouped together to form a contiguous tower of meta-modules. These towers

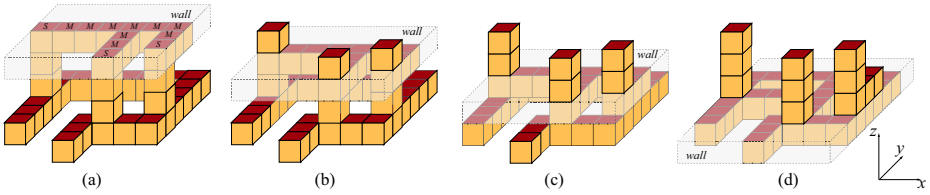


Fig. 5. The 3D-COMBING algorithm. (a) Meta-modules labeled M form one F-shaped connected component. (b, c, d) Robot configuration after (1, 2, 3) algorithm iterations. (d) Final terrain configuration.

extend in the z^+ direction, rest on an arbitrarily-shaped, connected base layer (in the xy-plane), and are attached only to the base layer.

Lemma 4. *A 3D robot can transform into a 3D terrain in place in $O(n)$ parallel steps and a total of $O(n)$ atom operations.*

3D Terrain to 3D Comb. A 3D Terrain I is reconfigured into a 3D comb by applying the 2D-COMBING algorithm of Sect. 3.1 to its base layer, thus reconfiguring the base layer into a 2D comb. As the base meta-modules move during the reconfiguration, they carry along the towers resting on top. If $B(I)$ is the base of I, then a call to 2D-COMBING($B(I)$) using the SLIDE operation that carries towers (see Fig. 1c) accomplishes this. After this second combing pass, the resulting 3D comb robot consists of a 2D comb in the xy-plane (call this the xy-comb), and each tooth and its handle module in the xy-comb form the handle of a comb with teeth extending up in the z direction (call these the z-combs). We immediately have the following result.

Lemma 5. *A 3D terrain can transform into a 3D comb in place in $O(n)$ parallel steps and a total of $O(n)$ atom operations.*

3D Comb to 3D Common Comb. Given two 3D combs C_S and C_T, this section describes an algorithm to reconfigure C_S into the 3D common comb determined by C_S and C_T. Let $s(t)$ be the number of z-combs in C_S (C_T); equivalently, $s(t)$ is the handle length of C_S's (C_T's) xy-comb. We assume C_S (C_T) is positioned with the handle of its xy-comb starting at lattice coordinates $(1, 1, 1)$ and extending to $(s, 1, 1)$ ($(t, 1, 1)$). Let C_S^i be the z-comb of C_S in lattice position i, let S_j^i be the jth tooth of C_S^i, and let $|S_j^i|$ be the number of teeth meta-modules in tooth S_j^i (not counting the handle module at its base). Let h_S^i be the length of C_S^i's handle. All terms are defined analogously for combs C_T and C_U.

As in 2D, comb C_U is the union of C_S and C_T. Let u be the handle length of C_U's xy-comb. The common comb is a subset of C_U consisting of the u handle meta-modules in its xy-comb and its rightmost $m - u$ meta-modules. More precisely, for each z-comb C_U^i, $i = u \ldots 1$, append to a list I the handle

meta-modules $(i, 2, 1)$ to $(i, h_U^i, 1)$ of C_U^i, followed by the teeth meta-modules of C_U^i in descending order on their y coordinate (primary key) and increasing order on their z coordinate (secondary key). The first $m - u$ meta-modules of I are in the common comb.

Algorithm 3 describes in detail the process of converting C_S to the common comb. In Step 1, the algorithm converts each z-comb C_S^i to the 2D common comb determined by $C_U^i = C_S^i \cup C_T^i$ using Algorithm 2. Since C_S^i and C_T^i may not contain the same number of meta-modules, there may not be enough meta-modules in C_S^i to fill the entire handle of C_U^i, in which case C_S^i will become only a portion of the handle that starts with module $(i, 1, 1)$. See Fig. 6a.

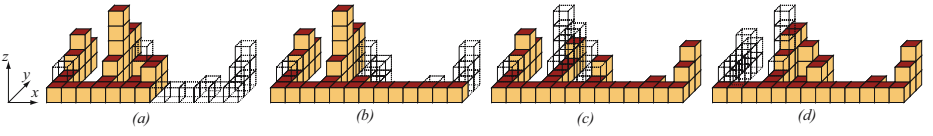


Fig. 6. (a) Solid meta-modules are C_S after each z-comb is converted to a common comb. C_U consists of the solid and the wireframe boxes. (b) C_S after extending its xy-comb handle to match that of C_U. (c) C_S during the execution of Step 4.3 of Algorithm 3, as it lengthens the teeth of C_S^7 by tunneling meta-modules from C_S^4. (d) The 3D common comb (solid boxes only).

Step 2 creates a queue, O, of meta-modules, in the order in which they will be used to fill meta-modules of C_U. Step 3 extends the length of C_S's xy-comb handle so that it matches the length of C_U's xy-comb handle. Figure 6b shows the results of this step. The order of the meta-modules in O ensures that each leg of the path is unattached to other meta-modules, thus allowing the TUNNEL move to be performed in $O(1)$ time. In Step 4, the teeth of each z-comb in C_S are lengthened to match the lengths of the corresponding teeth in C_U. Again, the order of the meta-modules in O ensures that each TUNNEL operation follows a path whose segments are not attached to other meta-modules, allowing $O(1)$ tunnel moves. A stage of Step 4 is illustrated in Fig. 6c, with Fig. 6d showing the resulting 3D common comb (solid meta-modules).

Lemma 6. *A 3D robot can transform into a common comb configuration in place in $O(n)$ parallel steps and a total of $O(n)$ atom operations.*

Overall 3D Reconfiguration Algorithm. The general algorithm to reconfigure any 3D m meta-module robot S to any 3D m meta-module target robot T consists of six stages: S reconfigures into 3D terrain I_S, then I_S reconfigures into 3D comb C_S, then C_S reconfigures into common comb C_{ST}, and finally the reverse moves reconfigure C_{ST} into C_T, C_T into I_T, and then I_T into T.

Theorem 2. *Any source robot can be reconfigured into any target robot in place in $O(n)$ parallel steps and a total of $O(n)$ atom operations.*

Algorithm 3 3D-Comb-To-Common-Comb Algorithm(C_S, C_U)

1. For $i = 1 \ldots s$
   1.1 2D-Comb-To-Common-Comb(C_S^i, C_U^i) (with combs parallel to the yz plane)
2. Let O be an empty queue
   For $i = s$ down to 1
   2.1 Append to O the teeth meta-modules of C_S^i, ordered by increasing y (primary key) and decreasing z (secondary key)
   2.2 Append to O all handle meta-modules of C_S^i except for module $(i, 1, 1)$, ordered by decreasing y
3. If $s < u$ then { extend the handle of C_S's xy-comb to length u }
   3.1 For $i = s + 1$ to u
   $oPos = O.dequeue()$
   In C_S, k-Tunnel($oPos$, $(i, 1, 1)$), for $k \in \{1, 2\}$
4. For $i = u$ down to 1 { fill in missing meta-modules of each z-comb }
   4.1 For $j = 1$ to $|C_S^i| - 1$ $O.dequeue()$ { remove meta-modules already in C_S^i }
   4.2 For $j = h_S^i + 1$ to h_U^i { lengthen handle of C_S^i }
   If $(O.size() == 0)$ exit
   $oPos = O.dequeue()$
   In C_S, k-Tunnel($oPos$, $(i, j, 1)$), for $k \in \{2, 3\}$
   4.3 For $j = h_S^i$ down to 1 {lengthen short teeth of C_S^i }
   For $k = |S_j^i| + 1$ to $|U_j^i|$
   If $(O.size() = 0)$ exit
   $oPos = O.dequeue()$
   In C_S, k-Tunnel($oPos$, (i, j, k)), for $k \in \{3, 4\}$

4 Distributed Implementation

Our centralized algorithms can be executed by the meta-modules in a synchronous, distributed fashion. The implementation must be synchronous since both the Slide and k-Tunnel moves require strict coordination of motion among the atoms in order to prevent collisions and disconnection of the robot. To synchronize the operations, we assume each atom/meta-module can count clock strikes modulo k, for any $k \in \mathbb{N}$.

The Combing algorithm is easily adaptable to the synchronous distributed model. During an initialization phase, each meta-module is sent its starting (x, y, z) location and the wall's starting position. Thereafter, each meta-module can determine its next move in $O(1)$ time using information on its current state (moving or stationary), or by polling adjacent meta-modules on their state. For example, each meta-module can determine its state by just checking if it is attached to a module below. The reverse of this algorithm can be made distributed in a similar way, sweeping the wall up instead of down.

The Comb-To-Common-Comb algorithms can also be distributed, albeit with some stronger requirements. First, the initial and final configurations S and T are communicated to each meta-module. In addition, each meta-module requires a more powerful processor on board. Specifically, we require that each meta-module can store information of size $O(n)$ and can run an algorithm of

complexity $O(n)$ in $O(n)$ time. These requirements are necessary because each meta-module must initially run the Comb-To-Common-Comb algorithm to precompute which operations it will perform on each clock strike, since local information alone is not enough to determine a meta-module's next operation. For example, meta-modules at the turn locations in the k-Tunnel operations must determine when they will be involved in such an operation in order to coordinate their actions. The reverse of this algorithm is similarly distributed.

Acknowledgments. We thank Thomas Hackl for his suggestion on how to reduce the size of the meta-modules [6]. We thank the other participants of the 2007 *Workshop on Reconfiguration* at the Bellairs Research Institute of McGill University for providing a stimulating research environment.

References

1. Rus, D., Vona, M.: Crystalline robots: Self-reconfiguration with compressible unit modules. Autonomous Robots 10(1), 107–124 (2001)
2. Vassilvitskii, S., Suh, Y.M.: A complete, local and parallel reconfiguration algorithm for cube style modular robots. In: Proc. of the IEEE Intl. Conf. on Robotics and Automation, pp. 117–122 (2002)
3. Butler, Z., Rus, D.: Distributed planning and control for modular robots with unit-compressible modules. The Intl. Journal of Robotics Research 22(9), 699–715 (2003)
4. Butler, Z., Fitch, R., Rus, D.: Distributed control for unit-compressible robots: Goal-recognition, locomotion and splitting. IEEE/ASME Trans. on Mechatronics 7(4), 418–430 (2002)
5. Suh, J.W., Homans, S.B., Yim, M.: Telecubes: Mechanical design of a module for self-reconfigurable robotics. In: Proc. of the IEEE Intl. Conf. on Robotics and Automation, pp. 4095–4101 (2002)
6. Hackl, T.: Personal communication (2007)
7. Aloupis, G., Collette, S., Damian, M., Demaine, E.D., Flatland, R., Langerman, S., O'Rourke, J., Ramaswami, S., Sacristán, V., Wuhrer, S.: Linear reconfiguration of cube-style modular robots (2007), `www.cs.siena.edu/papers/cbots_long.pdf`
8. Arkin, E., Bender, M., Mitchell, J., Polishchuk, V.: The snowblower problem. In: Proc. 7th Intl. Workshop on the Algorithmic Foundations of Robotics (2006)

Fast Message Dissemination in Random Geometric Ad-Hoc Radio Networks*

Artur Czumaj[1] and Xin Wang[2]

[1] Department of Computer Science, University of Warwick, Coventry CV4 7AL, United Kingdom
czumaj@dcs.warwick.ac.uk
[2] Department of Computer Science, New Jersey Institute of Technology, Newark, NJ 07102-1982, USA
xw37@oak.njit.edu

Abstract. We study the complexity of distributed protocols for the classical information dissemination problem of *distributed gossiping*. We consider the model of *random geometric networks*, one of the main models used to study properties of sensor and ad-hoc networks, where n points are randomly placed in a unit square and two points are connected by an edge/link if they are at at most a certain fixed distance r from each other. To study communication in the network, we consider the *ad-hoc radio networks* model of communication. We examine various scenarios depending on the local knowledge of each node in the networks, and show that in many settings distributed gossiping in asymptotically optimal time $\mathcal{O}(D)$ is possible, where D is the diameter of the network and thus a trivial lower bound for any communication.

1 Introduction

In this paper we study basic communication properties of random geometric networks as motivated by *mobile ad hoc networks* and *sensor networks*. Our main goal is to study under what conditions the dissemination of information can be performed efficiently, in particular, in time proportional to the diameter of the underlying network. We concentrate on the classical communication problem of *gossiping*: disseminating the messages in a network so that each node will receive messages from all other nodes.

Network model. We consider the standard model of *random geometric networks* [21]. A random geometric network $\mathcal{N} = (V, E)$ is an undirected graph with node set V corresponding to the set of transmitter-receiver stations placed *independently and uniformly at random* (i.u.r.) [1] in the unit square $[0, 1]^2$. The edges E of $\mathcal{N}$ connect specific pairs of nodes. We consider the *unit disc graph* model in

* Research supported in part by the *Centre for Discrete Mathematics and its Applications (DIMAP)*, University of Warwick.

[1] Another classical model assumes the points with *Poisson distribution* in $[0, 1]^2$. All our analyzes will work in that model too.

T. Tokuyama (Ed.): ISAAC 2007, LNCS 4835, pp. 220–231, 2007.

which for a given parameter r (called the *radius*) there is an edge between two nodes $p, q \in V$ if and only if the distance between p and q (denoted by $\mathsf{dist}(p, q)$) is smaller than or equal to r.

To study communication in the network, we consider the so-called *ad-hoc radio networks* model of communication [1,6,7,9,10,11,18]. We assume that all nodes have access to a global clock and work synchronously in discrete time steps called *rounds*. In radio networks the nodes communicate by sending messages through the edges of the network. In each round each node can either transmit the message to all its neighbors at once or can receive the message from one of its neighbors (be in the listening mode). A node x will receive a message from its neighbor y in a given round if and only if it does not transmit (is in the listening mode) and y is the only neighbor of x that is transmitting in that round. If more than one neighbor transmits simultaneously in a given round, then a collision occurs and no message is received by the node. In that case, we assume that the node cannot distinguish such a collision from the situation when none of its neighbors is transmitting. Furthermore, we assume the length of the message sending in one round is polynomial of n, and thus, each node can combine multiple messages into one.

Geometric models of knowledge. We consider the model of *ad-hoc networks*, in which the topology of the connections is not known in advance. In general, the nodes do not know their positions nor they know the positions of their neighbors, and each node only knows its ID (a unique integer in $[1, n^\lambda]$ for an arbitrary constant λ; this assumption can be removed in the randomized algorithm), its initial message, and the number of the nodes n in $\mathcal{N}$. (Since in all our settings, the running time is polynomial in n (because D is polynomial in n), this assumption can be removed by the standard doubling technique, without change the asymptotic time complexity.)

In many applications, one can assume that the nodes of the network have some additional devices that allow them to obtain some basic (geometric) information about the network. The most powerful model assumes that each node has a low-power Global Position System (GPS) device, which gives the node its exact location in the system [13]. Since GPS devices are relatively expensive, GPS is often not available. In such situation, we consider a *range-aware* model, the model extensively studied in the context of localization problem for sensor networks [2]. In this model, the distance between neighboring nodes is either known or can be estimated by received signal strength (RSS) readings with some errors. We also consider another scenario, in which each node can be aware of the direction of the incoming signals, that is, to measure the angles between different neighbors [20].

Properties of random geometric networks. It is known that when $r < (1 - o(1)) \cdot \sqrt{\ln n/(\pi\, n)}$, the network is disconnected with high probability [14], and therefore gossiping is meaningless in that case. Therefore, in this paper we will always assume that $r \geq c \cdot \sqrt{\log n/n}$ for some sufficiently large constant c. This ensures that the network is connected with high probability and therefore gossiping is

feasible. With this assumption we can also make some further assumptions about the structure of the input network. And so, it is well known (cf. [21]) that such a random geometric network has *diameter* $D = \Theta(1/r)$ and the minimum and maximum degree is $\Theta(n\,r^2)$, where all these claims hold *with high probability*, that is, with probability at least $1 - 1/n^{\Omega(1)}$. Therefore, from now on, we shall *implicitly* condition on these events.

Related prior works. In the *centralized scenario*, when each node knows the entire network, Kowalski and Pelc [18] gave a centralized deterministic broadcasting algorithm running in $\mathcal{O}(D + \log^2 n)$ time and Gąsieniec et al. [11] designed a deterministic $\mathcal{O}(D + \Delta \log n)$-time gossiping algorithm, where D is the diameter and Δ the maximum degree of the network.

There has been also a very extensive research in the *non-centralized (distributed)* setting in ad-hoc radio networks, see, e.g., [3,7,12,17,18] and the references therein. In the model of *unknown topology networks*, randomized broadcasting can be performed in the optimal $\mathcal{O}(D \log(n/D) + \log^2 n)$ time [7,17]; fastest deterministic algorithm runs in $\mathcal{O}(n \log^2 D)$ time [7]. The fastest randomized algorithm for gossiping in directed networks runs in $\mathcal{O}(n \log^2 n)$ time [7]; fastest deterministic one runs in $\mathcal{O}(n^{4/3} \log^4 n)$ time [12]. For undirected networks, both broadcasting and gossiping have deterministic $\mathcal{O}(n)$-time algorithms [1,4].

Dessmark and Pelc [9] consider broadcasting in ad-hoc radio networks in a model of geometric networks. They consider scenarios in which all nodes either *know their own locations in the plane*, or the labels of the nodes within some distance from them. The nodes use disks of possibly different sizes to define their neighbors. Dessmark and Pelc [9] show that *broadcasting* can be performed in $\mathcal{O}(D)$ time.

Recently, the complexity of broadcasting in ad-hoc radio networks has been investigated in a *(non-geometric)* model of $G_{n,p}$ *random networks* by Elsässer and Gąsieniec [10], and Chlebus et al. [5], and in the model of random line-of-sight ad-hoc radio networks by Czumaj and Wang [8].

New contributions. In this paper we present a thorough study of basic communication primitives in *random geometric ad-hoc radio networks*. We study information dissemination in various models of random geometric ad-hoc radio networks and we demonstrate that in many scenarios, the random structure of these networks allows us to perform distributed gossiping in asymptotically optimal time $\mathcal{O}(D)$.

We begin with the most restrictive model of local knowledge, the *unknown topology model*. In this model, the nodes have no global nor local information about the structure of the network. Still, we show that it is possible to perform distributed randomized gossiping in $\mathcal{O}(n\,r^2 \log n + D)$ time, with high probability. This is the first asymptotically optimal algorithm for random geometric ad-hoc radio unknown topology networks with $r \leq \mathcal{O}((n \log n)^{-1/3})$, in which case the running time is $\mathcal{O}(D)$.

Next, we consider deterministic distributed algorithms in three models in which the nodes have some geometric local information about the network. In

the first model we consider, if a node communicates with another node, then he is able to determine the *distance* to the node with which he communicates. In the next model, each node is able to determine *directions* to all its neighbors. Finally we consider the most powerful model in which each node knows its own *position* in $[0,1]^2$. The first two models are fairly similar and for them we design a distributed deterministic algorithms complete gossiping in optimal $\mathcal{O}(D)$ time, assuming $r \leq \mathcal{O}(n^{-7/16} \log^{-5/16} n)$. The model in which each node knows its own location is more powerful and we use the techniques from [9] to get a $\mathcal{O}(D)$-time deterministic algorithm for even larger range of r, $r \leq \mathcal{O}(n^{-2/5} \log^{-1/5} n)$.

For majority of applications of random geometric ad-hoc radio networks the underlying networks are sparse or are aimed to be as sparse as possible. Therefore, even though we present our algorithms to work for all values of $r \geq c\sqrt{\log n/n}$, our main focus is on networks with small values of r, just a little above connectivity threshold. For such networks, our algorithms have asymptotically optimal running times for a large range of the parameter r.

2 Preliminaries

For any node v, define $N(v)$ to be the set of nodes that are reachable from v in one hop, $N(v) = \{u \in V : \mathsf{dist}(v,u) \leq r\}$, where $\mathsf{dist}(v,u)$ is the Euclidean distance between v and u. Any node in $N(v)$ is called a *neighbor* of v, and set $N(v)$ is called the *neighboring set* of v. For any $X \subseteq V$, let $N(X) = \bigcup_{x \in X} N(x)$. Define the *kth neighborhood* of a node v, $N^k(v)$, recursively as follows: $N^0(v) = v$ and $N^k(v) = N(N^{k-1}(v))$ for $k \geq 1$. The *strict kth neighborhood* of v, denoted by $SN^k(v)$, is defined as $SN^k(v) = N^k(v) \setminus N^{k-1}(v)$.

Strongly-selective families. Let k and m be two arbitrary positive integers with $k \leq m$. Following [3], a family $\mathcal{F}$ of subsets of $\{1,\ldots,m\}$ is called (m,k)-*strongly-selective* if for every subset $X \subseteq \{1,\ldots,m\}$ with $|X| \leq k$, for every $x \in X$ there exists a set $F \in \mathcal{F}$ such that $X \cap F = \{x\}$. It is known (see, e.g., [3]) that for every k and m, there exists a (m,k)-strongly-selective family of size $\mathcal{O}(k^2 \log m)$.

With the concept of strongly-selective families, we are now ready to proceed to the following lemma.

Lemma 1. *In random geometric networks, for any integer k, in **(deterministic)** time $\mathcal{O}(k \cdot n^2 \cdot r^4 \cdot \log n)$ all nodes can send their messages to all nodes in their kth neighborhood. The algorithm may fail with probability at most $1/n^2$ (where the probability is with respect to the random choice of a geometric network).*

Proof. The proof uses nowadays standard approach of applying selective families to broadcasting and gossiping in radio ad-hoc networks, see, e.g., [3]. □

3 Randomized Gossiping in Optimal $\mathcal{O}(D)$ Time

In this section, we present a simple randomized algorithm for broadcasting and gossiping problem in random geometric networks whose running time is asymptotically optimal for small values of r. We see our algorithm as an extension of

the classical broadcasting algorithm in networks due to Bar-Yehuda et al. [1] (see also [7]), which when applied to random geometric networks gives asymptotically optimal runtime for a more complex task of gossiping (for small r).

repeat
in each round, each node independently does:
the node transmits with probability $\frac{1}{n\,r^2}$

Theorem 1. *The algorithm above completes gossiping in a random geometric network after $\mathcal{O}(n\,r^2\,\log n + D)$ rounds with probability at least $1 - 1/n$. If $r \leq \mathcal{O}\left(\frac{1}{(n\log n)^{1/3}}\right)$, then the number of rounds is $\mathcal{O}(D)$.*

Before we proceed with the proof of Theorem 1, let us first introduce some basic notation. Let us divide the unit square into $16/r^2$ *blocks* (disjoint squares), each block with the side length of $r/4$. For a block B, we also use B to denote the set of nodes in block B; in this case, $|B|$ is the number of nodes in block B.

The following lemma follows easily from Chernoff bounds.

Lemma 2. *For every block B with probability at least $1-1/n^4$: (i) $\frac{n\,r^2}{32} \leq |B| \leq n\,r^2$, (ii) $|N(B)| \leq 20\,n\,r^2$.*

A *gossiping within a block* is the task of exchanging the messages between all the nodes in the block. Gossiping within a block B is completed if every node $v \in B$ receives a message from every other $u \in B$.

Lemma 3. *Gossiping within every block completes in $\mathcal{O}(n\,r^2\,\log n)$ steps with probability at least $1 - \frac{1}{n^2}$.*

Proof. Fix a node $v \in B$. In any single round, the probability that node v transmits and no other node from $N(B) \setminus \{v\}$ transmits is at least $\frac{1}{n\,r^2}\,(1 - \frac{1}{n\,r^2})^{|N(B)\setminus\{v\}|} \geq \frac{1}{n\,r^2}\,(1 - \frac{1}{n\,r^2})^{20\,n\,r^2} \geq \frac{1}{n\,r^2}\,e^{-40}$. Hence, in any single step, v will send its message to all other nodes in block B with probability at least $\frac{1}{e^{40}\,n\,r^2}$. After τ steps, v sends its message to all other nodes in block B with probability at least $1-(1-\frac{1}{e^{40}\,n\,r^2})^{\tau}$. Hence, by the union bound, the probability that the gossiping within every single block will be completed after τ steps is greater than or equal to $1 - n \cdot (1 - \frac{1}{e^{40}\,n\,r^2})^{\tau}$. By choosing an appropriate large value of $\tau = \mathcal{O}(n\,r^2\,\log n)$, this probability will be greater than $1-\frac{1}{n^2}$, as needed.

At any time step t, let $M_t(v)$ be the set of messages currently held by node v. For any block B, let $M_t(B)$ denote the set of common messages that are currently held by all nodes of B, that is, $M_t(B) = \bigcap_{v\in B} M_t(v)$.

Lemma 4. *Let B and B' be two adjacent blocks and suppose that the gossiping within block B has been completed. Then, for any t, $M_t(B) \cup M_t(B') \subseteq M_{t+1}(B')$ with constant probability.*

Proof. By Lemma 2, $|N(B')| \leq 20\, n\, r^2$ and $|B| \geq n\, r^2/32$ with high probability. Therefore, conditioned on these two inequalities, with probability $p \geq |B| \cdot \frac{1}{n\, r^2} \cdot (1-\frac{1}{n\, r^2})^{|N(B')|} \geq n\, r^2/32 \cdot \frac{1}{n\, r^2} \cdot (1-\frac{1}{n\, r^2})^{20\, n\, r^2}$, among all nodes in $N(B')$, there is exactly one node in B that transmits at a given time step. For n big enough, p is greater than some positive constant c'. This yields the claim.

Now, we are ready to complete the proof of Theorem 1. Let us focus on two blocks B and B'. By Lemma 3, gossiping within every block will be completed after the first $\mathcal{O}(n\, r^2 \log n)$ steps w.h.p. For fixed blocks B and B', there is always a sequence of blocks $B = B_1, B_2, \ldots, B_k = B'$, such that B_i and B_{i+1} are adjacent for any $1 \leq i \leq k-1$, and that $k \leq 8/r$. By Lemma 4, after each step, B_i will send its message $M_t(B_i)$ to B_{i+1} with probability at least c', where c' is a positive constant promised by Lemma 4. Therefore, by a simple application of known concentration results for random variables with negative binomial distribution, after $\mathcal{O}(k/c' + \log n) = \mathcal{O}(D + \log n)$ steps, all the messages from B will be successfully transmitted to B' with probability at least $1 - 1/n^4$. By applying the union bound on all pairs of blocks, we conclude that gossiping is completed with probability at least $1 - 1/n^2$. □

Randomized broadcasting. Our analysis in Theorem 1 can be improved for the broadcasting problem, where for every r we can obtain the running time of $\mathcal{O}(D + \log n)$. (Details deferred to the full version.)

4 Deterministic Distributed Algorithm: Knowing Distances Helps

In this section, we assume that $c\sqrt{\log n/n} \leq r \leq \mathcal{O}(n^{-7/16} \log^{-5/16} n)$ and show that the gossiping in random geometric networks can be done optimally in time $\mathcal{O}(D)$ in the *range-aware* model.

Building a local map. The key property of our model that we will explore in the optimal gossiping algorithm is that by checking the inter-point distances, we can create a "map" with relative locations of the points. Indeed, if for three points u, v, w, we know their inter-points distances, then if we choose u to be the origin (that is, has location $(0, 0)$), we can give relative locations of the other two points v and w. (The relative location is not unique because there are two possible locations, but by symmetry, any of these two positions will suffice for our analysis.) We will show later that with such a map, the gossiping task can be performed optimally. (Let us point out that even with local coordinate system, the global consistent position information is still unavailable.)

The following lemma easily follows from Lemma 1 and the discussion above.

Lemma 5. *After $\mathcal{O}(D)$ communication steps, all nodes $u \in \mathcal{N}$ can learn $\mathsf{dist}(u, v)$ for any node $v \in N^\tau(u)$, where $\tau = \lceil 1/(n^2\, r^5 \log n) \rceil$. (This algorithm may fail with probability at most $1 - 1/n^3$.)*

This lemma implies not only that $u \in \mathcal{N}$ can learn $\mathsf{dist}(u,v)$ for any node $v \in N^\tau(u)$, but also that it can set up its own local map of the nodes in $N^\tau(u)$. From now on, we will proceed with $\tau = \lceil 1/(n^2 r^5 \log n)\rceil$.

Boundary and corner nodes. In our algorithm we consider two special types of nodes: *boundary nodes* and *corner nodes*.

If a node u observes that there is a sector with angle $\pi/2$ that is centered at u so that every neighbor of u in that sector is at a distance at most $r/\sqrt{2}$, then u marks itself as a *boundary node*. It is easy to see that with high probability, a node is a boundary node only if its distance to the boundary of $[0,1]^2$ is less than r, and also every node which is at a distance at most $r/2$ from the boundary is a boundary node. Similarly, a node u marks itself as a *corner node* if there is a line going through u for which all neighbors of u that are on one side of the line have distance at most $r/2$ from u. It is easy to see that with high probability, every corner node is at a distance at most r from a corner of $[0,1]^2$ and every node that is at a distance at most $r/4$ from a corner of $[0,1]^2$ is a corner node.

Next, we select one *corner representative node* for each corner of $[0,1]^2$. It can be done easily by Lemma 1.

Transmitting along boundary nodes. Now, we will show that the gossiping among boundary nodes can be performed in optimal $\mathcal{O}(D)$ time.

The process of the gossiping among the boundary nodes is initialized by the four corner representative nodes. Each corner representative node u checks its map of the nodes in $N^\tau(u)$ and selects two farthest boundary nodes, one for each boundary. Then, it sends a message to these two nodes with the aim of transmitting its message to the two neighboring corner representative nodes

The process of sending messages to the corners works in *phases*. In each phase, there are up to eight pairs of nodes ϖ_i^j and ϖ_{i+1}^j such that ϖ_i^j wants to transmit a message to ϖ_{i+1}^j, with both ϖ_i^j and ϖ_{i+1}^j being boundary nodes and $\varpi_{i+1}^j \in N^\tau(\varpi_i^j)$. At the beginning of the phase, ϖ_i^j checks its local map and finds a path P_{ij} from ϖ_i^j to ϖ_{i+1}^j of length at most τ. Then, it transmits to its neighbors and request that only the first node on P_{ij} will transmit the message to ϖ_{i+1}^j. Then, the first node on P_{ij} will transmit to its neighbors and will request that only the second neighbor on P_{ij} will transmit, and so on, until ϖ_{i+1}^j will receive the message. Once ϖ_{i+1}^j received a message, it sends back an acknowledgement to ϖ_i^j that the message has been delivered. The algorithm for sending an acknowledgement is a reverse of the algorithm for transmitting a message from ϖ_i^j to ϖ_{i+1}^j. The last step of each phase is to establish the next nodes ϖ_{i+2}^j. If ϖ_i^j sent a message to ϖ_{i+1}^j then ϖ_{i+1}^j checks its map and selects as ϖ_{i+2}^j a node in $\varpi_{i+2}^j \in N^\tau(\varpi_{i+1}^j)$ that is farthest from ϖ_i^j. As an exception, if one of the corner representative nodes is in $N^\tau(\varpi_{i+1}^j) \setminus \{\varpi_i^j\}$, then this corner representative node is selected as ϖ_{i+2}^j and then the process stops, i.e., ϖ_{i+3}^j will not be selected.

Obviously, if there are no transmission conflicts between the eight pairs ϖ_i^j and ϖ_{i+1}^j, then each phase can be performed in 2τ communication steps (including sending the acknowledgements). The only way of having a transmission conflict is that two pairs ϖ_i^j and ϖ_{i+1}^j, and $\varpi_i^{j'}$ and $\varpi_{i+1}^{j'}$, are transmitting along the same boundary and that in this phase $N^\tau(\varpi_i^j) \cap N^\tau(\varpi_i^{j'}) \neq \emptyset$. If this happen, then the nodes ϖ_i^j and $\varpi_i^{j'}$ may not obtain an acknowledgement. In this case, both ϖ_i^j and $\varpi_i^{j'}$ repeat the process of transmitting their messages to ϖ_{i+1}^j and $\varpi_{i+1}^{j'}$, respectively, using the selector approach from Lemma 1 that ensures that the phase will be completed in $\mathcal{O}(\tau \cdot n^2\, r^4\, \log n) = \mathcal{O}(D)$ communication steps.

Let ϱ_1 and ϱ_2 be two adjacent corner representative nodes, and $(\varrho_2, \varpi_1^j, \varpi_2^j, \varpi_3^j, \ldots \varrho_1)$ be a sequence of nodes initialized by ϱ_2 in the process described before. It is easy to see that: *(i)* ϱ_1 receives all the messages of $\varrho_2, \varpi_1^j, \varpi_2^j, \varpi_3^j, \ldots$, *(ii)* ϱ_2 sends its message to all nodes in $\varpi_1^j, \varpi_2^j, \varpi_3^j, \ldots \varrho_1$, and *(iii)* for any boundary node v, there is a ϖ_i^j such that $v \in N^\tau(\varpi_i^j)$ (which holds because of the way we pick ϖ_i^j).

Therefore, each corner representative node will receive all messages from the boundary nodes of its incident boundaries. If we repeat this process again, then each corner representative node will receive the messages of *all* boundary nodes. If we repeat this process once again, then all ϖ_i^j nodes will receive the messages from all boundary nodes. If we now apply the approach from Lemma 1, then each boundary node will receive a message from at least one ϖ_i^j, and hence it will receive messages from all boundary nodes.

By our comments above, if there is no conflict in a phase, then the phase is completed in 2τ communication steps, but if there is a conflict, then the number of communication steps in the phase is $\mathcal{O}(\tau\, n^2\, r^4\, \log n)$. Now, we observe that if a corner representative node originates a transmission that should reach another corner representative node, then there will be at most a constant number of phases in which there will be a conflict. Therefore, the total running time for this algorithm is $\mathcal{O}(\tau \cdot D/\tau) + \mathcal{O}(\tau\, n^2\, r^4\, \log n) = \mathcal{O}(D)$.

Lemma 6. *The algorithm above completes gossiping among all boundary nodes in $\mathcal{O}(D)$ time.*

Gossiping via transmitting along almost parallel lines. Let ϱ be the corner representative node with the smallest ID. Let ϱ^* be the corner representative node that shares the boundary with ϱ (there are two such nodes) and that has the smaller ID. Let ϱ select $\mathcal{O}(D/\tau)$ boundary nodes $\varsigma_1, \varsigma_2, \ldots$ such that *(i)* $\varsigma_{i+1} \in N^{\lfloor \tau/4 \rfloor}(\varsigma_i)$ for every i, and *(ii)* $\varsigma_j \notin N^{\lfloor \tau/32 \rfloor}(\varsigma_i)$ for every i, j, $i \neq j$. It is easy to see that such a sequence exists, and that ϱ is able to determine it because after Lemma 6, ϱ knows all boundary nodes and their τ neighbors. Next, ϱ informs all boundary nodes about its choice using the process from the previous section. We now present an algorithm in which all the nodes ς_i will originate a procedure ***Straight-line transmission*** aiming at disseminating the information contained by these nodes along a line orthogonal to the boundary shared by ϱ and ϱ^*.

There are a few problems with this approach that we need to address. First of all, we do not know the boundary of the unit square and instead, the goal will be to consider lines orthogonal to the line $\mathcal{L}$ going through ϱ and ϱ^*. The location of $\mathcal{L}$ can be determined from the local map known to all the boundary nodes. Notice that since the angle between the boundary of $[0,1]^2$ and $\mathcal{L}$ is at most $\mathcal{O}(r)$, $\mathcal{L}$ is a good approximation of the boundary of $[0,1]^2$. Next, we observe that we will not be able to do any transmissions along any single line because our network $\mathcal{N}$ does not contain three collinear nodes with high probability. Therefore, our process will need to proceed along an approximate line. We begin with the following lemma that will help us quantify the angle between the perfect line we want to transmit along and the line along which we will actually transmit. The lemma easily follows from the Chernoff Bound.

Lemma 7. *Let* $\tau = \lceil 1/(n^2\, r^5\, \log n)\rceil$ *and* $r \leq \mathcal{O}\left(\frac{1}{n^{7/16}\, \log^{5/16} n}\right)$. *Let* u *be a node in* $\mathcal{N}$ *and let* ℓ_u *be any ray (half-line) starting at* u. *If all points* $q \in \ell_u$ *with* $\mathsf{dist}(u,q) \leq \tau \cdot r$ *are contained in* $[0,1]^2$ *then with high probability there is a node* $w \in N^{\lfloor \tau/4 \rfloor}(u) \setminus N^{\lceil \tau/32 \rceil}(u)$ *such that* $|\measuredangle(\ell_u u w)| \leq O(\tau^2 r^2)$.

Now, we use Lemma 7 to design a scheme that allows a point to transmit a message along an approximate line. Our procedure Straight-line transmission(s,μ,ℓ) aims at transmitting a message μ from node s along (approximately) line ℓ_s, $s \in \ell_s$, so that all nodes that are close to ℓ_s will receive the message μ.

In Straight-line transmission(s,μ,ℓ_s), the node s initiates sending its message μ along the line ℓ_s. The transmission process is performed in *phases*; each phase consists of sending a message from a node ϖ_i to another node ϖ_{i+1} such that ℓ_s is approximately equal to the line going through ϖ_i and ϖ_{i+1}, and $\varpi_{i+1} \in N^{\lfloor \tau/4 \rfloor}(\varpi_i) \setminus N^{\lfloor \tau/32 \rfloor}(\varpi_i)$. The nodes ϖ_i are determined recursively. Initially, $\varpi_0 = s$ and ϖ_1 is the node $q \in N^{\lfloor \tau/4 \rfloor}(s) \setminus N^{\lfloor \tau/32 \rfloor}(s)$ for which $|\measuredangle(\ell_s s q)|$ is minimized. If $i \geq 1$ and ϖ_i is determined, then *(i)* if $N^\tau(\varpi_i) \setminus \{\varpi_{i-1}\}$ contains a boundary node then ϖ_{i+1} is undefined and the process is stopped; *(ii)* otherwise, ϖ_{i+1} is selected to be $u \in N^{\lfloor \tau/4 \rfloor}(\varpi_i) \setminus N^{\lfloor \tau/32 \rfloor}(\varpi_i)$ for which $|\measuredangle(\varpi_{i-1}\varpi_i\varpi_{i+1}) - \pi|$ is minimized. Since ϖ_i knows the locations of all nodes in $N^\tau(\varpi_i)$, ϖ_i is able to select ϖ_{i+1} using its local map. Observe that by Lemma 7, for every node ϖ_i, $i \geq 1$, we have $|\measuredangle(\varpi_{i-1}\varpi_i\varpi_{i+1}) - \pi| \leq \mathcal{O}(\tau^2\, r^2)$, with high probability. Next, since $\mathsf{dist}(\varpi_i\varpi_{i+1}) = \Theta(\tau \cdot r)$, we conclude that the last representative ϖ_i will have index $\mathcal{O}(\frac{1}{\tau\, r})$. Hence, for every $i \geq 2$, we have $|\measuredangle(\ell_s u \varpi_i)| \leq \mathcal{O}(\frac{1}{\tau\, r} \cdot \tau^2\, r^2) = \mathcal{O}(\tau\, r)$ with high probability. The running time of each phase of Straight-line transimmision is $O(\tau)$. So the running time of Straight-line transmission is $\mathcal{O}(\tau \cdot \frac{1}{\tau\, r}) = \mathcal{O}(D)$.

We will run multiple calls to Straight-line transmission(s,μ,ℓ_s) with s being the nodes ϱ, ϱ^*, and $\varsigma_1, \varsigma_2, \ldots$, as defined earlier and with line ℓ_s being the line going through s that is orthogonal to the line $\mathcal{L}$ (which is the line going through ϱ and ϱ^*).

It is easy to see in random geometric networks, if u is the strict $k^{(th)}$ neighbor of v, then $dist(u,v) \geq k\, r/4$ with high probability. Hence the distance between any of the points ϱ, ϱ^*, and $\varsigma_1, \varsigma_2, \ldots$ is at least $\Omega(\tau\, r)$, so are the distance between

the lines ℓ_s. On the other hand, as we argued above, every procedure Straight-line transmission(s, μ, ℓ_s) is sending messages only among the nodes that are at distance at most $O(\tau\, r)$ from the line ℓ_s, where this claim holds with high probability. Therefore, in particular, the communication in the calls to Straight-line transmission(s, μ, ℓ_s) will be done without any interference between the calls, with high probability (to avoid collisions between adjacent lines we interleave the transmissions in adjacent lines, yielding a $\mathcal{O}(1)$ factor slow-down).

Lemma 8. *All calls to Straight-line transmission(s, μ, ℓ_s) with s being ϱ, ϱ^*, and $\varsigma_1, \varsigma_2, \ldots$ can be completed in $\mathcal{O}(D)$ communication steps, with high probability.*

Observe that while running the procedures Straight-line transmission, each node that is transmitting can include in its message also all the knowledge it contains at a given moment. Therefore, in particular, each last node ϖ_k will receive all the messages collected on its path from s.

Next, let us observe that for every node q in the network $\mathcal{N}$ either q has been selected as one of the nodes ϖ_i in one of the calls to Straight-line transmission or one of the nodes in $N^\tau(q)$ has. Indeed, since the distance between the adjacent lines ℓ_s is at most $\lfloor \tau/4 \rfloor \cdot r$, for each point $q \in \mathcal{N}$ there is a line ℓ_s with $\mathsf{dist}(q, \ell_s) \leq \lfloor \tau/4 \rfloor \cdot r/2$. Therefore, there will be at least one node ϖ_i for Straight-line transmission(s, μ, ℓ_s) with $\mathsf{dist}(q, \varpi_i) \leq \tau \cdot r/2$. This yields $\varpi_i \in N^\tau(q)$ with high probability. Because of this, if all nodes $u \in \mathcal{N}$ know the messages from all nodes in $N^\tau(u)$, then after completing the calls to Straight-line transmission, for each node $u \in \mathcal{N}$ there will be at least one boundary node that received the message of u.

If we do gossiping among the boundary nodes once again, all the boundary nodes will have the messages from all the nodes in $\mathcal{N}$. Next, we run again Straight-line transmission(s, μ, ℓ_s) with s being ϱ, ϱ^*, and $\varsigma_1, \varsigma_2, \ldots$ as defined above. Then, all the nodes ϖ_i will obtain the messages from all nodes in $\mathcal{N}$. Finally, since each $q \in \mathcal{N}$ has in its τ-neighborhood a node ϖ_i, we can apply Lemma 1 to ensure that all nodes in $\mathcal{N}$ will receive the messages from all other nodes in $\mathcal{N}$.

Theorem 2. *Let $c\sqrt{\log n/n} \leq r \leq \mathcal{O}\left(\frac{1}{n^{7/16}\, \log^{5/16} n}\right)$. In the range-aware model, there is a deterministic distributed algorithm that completes gossiping in a random geometric network can be completed in deterministic time $\mathcal{O}(D)$. The algorithm may fail with probability at most $1/n^2$.*

5 Deterministic Distributed Algorithm: Knowing Angles Helps

One can modify the algorithm from Theorem 2 to work in the scenario in which a node cannot determine the distance between its neighboring node but instead, it is able to determine the relative direction where the neighbor is located.

Theorem 3. *Let* $c\sqrt{\log n/n} \leq r \leq \mathcal{O}\left(\frac{1}{n^{7/16}\,\log^{5/16} n}\right)$. *If each node receiving the message is able to determine the relative direction from which the message arrives, then gossiping in a random geometric network can be completed in deterministic time* $\mathcal{O}(D)$. *The algorithm may fail with probability at most* $1/n^2$.

The algorithm is essentially the same as that described in Section 4 with two differences. First of all, now the local map of a node does not have the exact distances but it may be re-scaled. That is, using the same approach as presented in Section 4, each node can build its local map where all the angles in the map are the actual angles between the points, but only the distances may be re-scaled. Secondly, we need another approach to determine if a node is a boundary node or it is a corner node. This can be done by comparing the density of the neighborhoods of the nodes (Details deferred to the full version.)

6 Deterministic Distributed Algorithm: Knowing Locations Helps

We consider also the gossiping problem in random geometric networks in the power model, where each node knows its geometric position in the unit square. In such model, Dessmark and Pelc [9] give a deterministic algorithm for *broadcasting* that (in our setting) runs in $\mathcal{O}(D)$ time. We can prove a similar result for *gossiping* by extending the preprocessing phase from [9] and use an appropriate strongly-selective family to collect information about the neighbors of each point (Details deferred to the full version.)

Theorem 4. *If every input node knows its location* $[0,1]^2$, *then there is a deterministic algorithm that completes gossiping in a random geometric network in time* $\mathcal{O}(n^2\, r^4\, \log n + 1/r)$. *In particular, if* $r \leq \mathcal{O}\left(\frac{1}{n^{2/5}\,\log^{1/5} n}\right)$ *then the running time is* $\mathcal{O}(D)$. *The algorithm may fail with probability at most* $1/n^2$.

7 Conclusions

In this paper we presented the first thorough study of basic communication aspects in random geometric ad-hoc radio networks. We have shown that in many scenarios, the random structure of these networks (which often may model well realistic scenarios from sensor networks) allows us to perform communication between the nodes in the network in asymptotically optimal time $\mathcal{O}(D)$, where D is the diameter of the network and thus a trivial lower bound for any communication. This is in contrast to arbitrary ad-hoc radio networks, where deterministic bounds of $o(n)$ are unattainable.

Our study shows also that while there is a relatively simple optimal randomized gossiping algorithm and a deterministic one when the nodes have knowledge about their locations in the plane, the other scenarios are more complicated. In particular, we do not know if $\mathcal{O}(D)$-time deterministic gossiping is possible in the unknown topology model.

References

1. Bar-Yehuda, R., Goldreich, O., Itai, A.: On the time-complexity of broadcast in multi-hop radio networks: An exponential gap between determinism and randomization. Journal of Computer and System Sciences 45(1), 104–126 (1992)
2. Capkun, S., Hamdi, M., Hubaux, J.-P.: GPS-free positioning in mobile ad hoc networks. Cluster Computing 5(2), 157–167 (2002)
3. Clementi, A.E.F., Monti, A., Silvestri, R.: Distributed broadcast in radio networks of unknown topology. Theoretical Computer Science 302, 337–364 (2003)
4. Chlebus, B.S., Gąsieniec, L., Gibbons, A., Pelc, A., Rytter, W.: Deterministic broadcasting in ad hoc radio networks. Distributed Computing 15(1), 27–38 (2002)
5. Chlebus, B.S., Kowalski, D.R., Rokicki, M.A.: Average-time coplexity of gossiping in radio networks. In: Flocchini, P., Gkasieniec, L. (eds.) SIROCCO 2006. LNCS, vol. 4056, pp. 253–267. Springer, Heidelberg (2006)
6. Chrobak, M., Gąsieniec, L., Rytter, W.: Fast broadcasting and gossiping in radio networks. Journal of Algorithms 43(2), 177–189 (2002)
7. Czumaj, A., Rytter, W.: Broadcasting algorithms in radio networks with unknown topology. Journal of Algorithms 60(2), 115–143 (2006)
8. Czumaj, A., Wang, X.: Communication problems in random line-of-sight ad-hoc radio networks. In: Proc 4th Symposium on Stochastic Algorithms, Foundations, and Applications (SAGA), pp. 70–81 (2007)
9. Dessmark, A., Pelc, A.: Broadcasting in geometric radio networks. Journal of Discrete Algorithms 5(1), 187–201 (2007)
10. Elsässer, R., Gąsieniec, L.: Radio communication in random graphs. Journal of Computer and System Sciences 72, 490–506 (2006)
11. Gąsieniec, L., Peleg, D., Xin, Q.: Faster communication in known topology radio networks. In: Proc. 24th PODC, pp. 129–137 (2005)
12. Gąsieniec, L., Radzik, T., Xin, Q.: Faster deterministic gossiping in directed ad-hoc radio networks. In: Hagerup, T., Katajainen, J. (eds.) SWAT 2004. LNCS, vol. 3111, pp. 397–407. Springer, Heidelberg (2004)
13. Giordano, S., Stojmenovic, I.: Position based ad hoc routes in ad hoc networks. In: Handbook of Ad Hoc Wireless Networks. ch. 16, pp. 1–14 (2003)
14. Gupta, P., Kumar, P.R.: Critical power for asymptotic connectivity in wireless networks. In: Stochastic Analysis, Control, Optimization and Applications: A Volume in Honor of W. H. Fleming, pp. 547–566 (1998)
15. Gupta, P., Kumar, P.R.: The capacity of wireless networks. IEEE Transactions on Information Theory IT-46, 388–404 (2000)
16. Ko, Y.B., Vaidya, N.H.: Location aided routing (LAR) in mobile adhoc networks. In: Proc 4th Annual ACM/IEEE International Conference on Mobile Computing and Networking (MOBICOM), pp. 66–75 (1998)
17. Kowalski, D., Pelc, A.: Broadcasting in undirected ad hoc radio networks. Distributed Computing 18(1), 43–57 (2005)
18. Kowalski, D., Pelc, A.: Optimal deterministic broadcasting in known topology radio networks. Distributed Computing 19(3), 185–195 (2007)
19. Li, X., Shi, H., Shang, Y.: A partial-range-aware localization algorithm for Ad-hoc wireless sensor networks. In: Proc. 29th Annual IEEE International Conference on Local Computer Networks (2004)
20. Nasipuri, A., Li, K., Sappidi, U.R.: Power consumption and throughput in mobile ad hoc networks using directional antennas. In: Proc. 11th IEEE International Conf. on Computer Communications and Networks (ICCCN), pp. 620–626 (2002)
21. Penrose, M.D.: Random Geometric Graphs. Oxford University Press, UK (2003)

Sensor Network Gossiping or How to Break the Broadcast Lower Bound*

Martín Farach-Colton and Miguel A. Mosteiro

Department of Computer Science, Rutgers University, Piscataway, NJ 08854, USA
{farach,mosteiro}@cs.rutgers.edu

Abstract. Gossiping is an important problem in Radio Networks that has been well studied, leading to many important results. Due to strong resouce limitations of sensor nodes, previous solutions are frequently not feasible in Sensor Networks. In this paper, we study the gossiping problem in the restrictive context of Sensor Networks. By exploiting the geometry of sensor node distributions, we present reduced, optimal running time of $O(D + \Delta)$ for an algorithm that completes gossiping with high probability in a Sensor Network of unknown topology and adversarial wake-up, where D is the diameter and Δ the maximum degree of the network. Given that an algorithm for gossiping also solves the broadcast problem, our result proves that the classic lower bound of [16] can be broken if nodes are allowed to do preprocessing.

1 Introduction

The *Radio Network* is a simplified abstraction of a radio-communication network. The question of how to diseminate information within such networks has led to different well-studied problems. Those problems differ on the number of network nodes holding messages to transmit, the number of different messages to be transmitted and the number of nodes that must receive those messages. A *message* is the piece of information that a node holds which must be distributed to other nodes. For settings where all nodes in the network must receive all the messages, the problems studied differ in the number of nodes that hold those messages, as follows. When k arbitrary nodes have a message the problem is known as k-*selection* [14]. If $k = 1$ the problem is called *Broadcast* [1,16], and if $k = n$, the size of the network, it is called *Gossiping* [17,5].

We study the gossiping problem in Sensor Networks, a network where n *sensor nodes* with processing, communication and sensing capabilities are distributed randomly in an area of interest in order to self-organize as a radio-communication network. Sensor Networks are expected to be used to gather information over large remote areas in hostile environments. Sensor nodes have to transmit such

* This research was supported in part by NSF grants CCF0621425, CCF 05414009, CCF 0632838, and DIMACS, Center for Discrete Mathematics & Theoretical Computer Science, grant numbered NSF CCR 00-87022.

T. Tokuyama (Ed.): ISAAC 2007, LNCS 4835, pp. 232–243, 2007.

information to distinguished nodes called *sinks*. The problem becomes challenging because sensor nodes are subject to strict resource limitations. Since the identity or the location of those sinks are frequently assumed to be unknown to the sensor nodes, gossiping is an important problem and its solution would yield an efficient communication primitive in this setting.

Gossiping in Radio Networks is a well-studied problem for which important results have been obtained. At least one of the following two crucial assumptions is present in all of these results. It is frequently assumed that the size of a message transmitted in one step is bounded only by the size of all the messages in the network, thus, nodes can pad their own message with all messages received and then re-transmit. In addition, a usual assumption is that either nodes start simultaneously or a global clock is available. In the Weak Sensor Model [10], a harsh and comprehensive model that summarizes the literature on sensor node restrictions, none of these assumptions are feasible because memory is limited, wake-up schedule is adversarial and no global clock is available.

Although in the asymptotic analysis $n \to \infty$, the memory limitation can be relaxed in practice when the magnitude of n is expected to be very large but bounded. However, since the deployment is produced in hostile or remote large areas, a global clock or a synchronous start is too strong of an assumption. Thus, we study gossiping in a relaxed version of the Weak Sensor Model where memory size is bounded only by a linear number of messages. We leave open the question of how to improve these results for constant-bounded memory size.

Related Work. Bar-Yehuda, Israeli and Itai [2] presented a randomized algorithm for Radio Networks with a topology modeled by an undirected graph, or *symmetric networks*, that completes gossiping in $O(n \log^2 n)$[1] on average. Briefly, their technique, used previously in [4] and later re-utilized in [12], is to build an underlying BFS tree to first collect all messages in the root node and later disseminate all of them to all nodes. In that paper, nodes know the identity of their neighbors, the size of the network n, and an upper bound on the maximum degree. Nodes may transmit and receive only $O(\log n)$ bit messages in synchronous time slots. However, they can store as many as needed. The same bound but with high probability[2] was proved in [5]. The algorithm relies on unbounded message size and global synchronism.

For directed graph topologies or *asymmetric networks*, Chrobak, Gąsieniec and Rytter [7] showed an upper bound of $O(n \log^3 n \log(n/\epsilon))$ with probability $1 - \epsilon$ and $O(n \log^4 n)$ in expectation. The main idea is to repeatedly run a limited broadcast that doubles the number of copies of each message in the network in each phase. Thus, unbounded message size is necessary as well as global synchronism.

[1] Througout this paper, log means $\log_2$ unless otherwise stated.

[2] Define *with high probability*, or *w.h.p.* for short, to mean with probability at least $1 - O(n^{-O(1)})$. We say that a parameterized event E_p occurs *with high probability* if for any constant $\gamma > 0$ there exists a valid choice of parameter p such that $Pr\{E_p\} \geq 1 - n^{-\gamma}$.

Using the same protocol, but improving the limited broadcast by adding randomization to it, Liu and Prabhakaran [17] reduced that upper bound by a logarithmic factor. More recently, Czumaj and Rytter [9] obtained a bound of $O(n \log^2 n)$ w.h.p. for this protocol by replacing the limited broadcast by a linear randomized broadcast where the probabilities are chosen with a special distribution. The model in all these results is a directed strongly connected graph where nodes have unique ID's in $\{1, \ldots, n\}$, work synchronously, and memory and messages are bounded only to the size of all messages.

Recently, Ravelomanana [20] studied the gossiping problem for the important class of networks with topology modeled by a random geometric graph, a model widely used in the Sensor Network area. The algorithm presented completes gossiping in $O(\sqrt{n} \log n)$ w.h.p. In a first stage, nodes obtain an ID and define a coloring in order to avoid collisions later in the gossiping phase. This algorithm is claimed to be optimal, but the lower bound used to prove it is the well-known result of Kushilevitz and Mansour [16] where nodes are not allowed to do anything before receiving the broadcast message.

Regarding deterministic solutions, Chrobak, Gąsieniec and Rytter [6] presented a $O(n^{3/2} \log^2 n)$ algorithm for asymmetric networks. The protocol makes use of selecting sequences to ensure non-colliding transmissions. In this model nodes have different ID's in $\{1, \ldots, n\}$, the topology is modeled by a directed graph, memory and messages are unbounded and global synchronism is assumed.

Results for the broadcast problem can be used as lower bounds for gossiping because the former can be solved using an algorithm for the latter. However, it should be noticed that, in order to prove lower bounds, broadcast protocols are defined leaving out solutions that include a pre-processing stage [16]. Bruschi and Del Pinto [3] proved a $\Omega(D \log n)$ lower bound, in a model where all nodes start simultaneously and nodes know their message history. More recently, Clementi, Monti and Silvestri [8] improved the lower bound to $\Omega(n \log D)$ in symmetric networks even if nodes are not synchronized. Kowalski and Pelc [15] constructed a class of graphs of diameter 4, such that every broadcasting algorithm requires time $\Omega(n^{1/4})$ on one of these graphs. The best general lower bound for randomized protocols is $\Omega(D \log(n/D))$ obtained by Kushilevitz and Mansour [16].

As for lower bounds for gossiping, Chlebus, Gąsieniec, Lingas and Pangourtzis [5] proved that any deterministic oblivious gossiping algorithm requires at least $n^2/2 - n/2 + 1$ steps to complete. In the same paper, for the important class of fair randomized protocols, i.e., protocols where all nodes use the same probability of transmission in the same time step, it was proved that for any integer $n \leq q \leq n^2/2$ there exists an asymmetric network such that the expected time to complete gossiping is $\Omega(q)$. More recently, Gąsieniec and Potapov [12] showed lower bounds of $\Omega(n^2)$ for asymmetric networks and $\Omega(n \log n)$ for symmetric networks. The topology of the construction used for the later can not be embeded in geometric graphs, therefore does not apply to Sensor Networks.

Our Results. We study the gossiping problem in a relaxed version of the Weak Sensor Model where memory size is bounded only by a linear number of messages. We present a randomized algorithm that, given a network of n nodes, with high

probability completes gossiping in $O(\Delta + D)$ time steps after the last node starts running the algorithm. Given that $\Omega(D)$ and $\Omega(\Delta)$ are lower bounds for this problem, this algorithm is optimal. This result improves over previous bounds in time efficiency and makes no assumptions about global synchronism. Rather, it exploits the geometry of the topology in Sensor Networks. Our result also shows that the classical broadcast lower bound of Kushilevitz and Mansour [16] can be broken if nodes are allowed to do some preprocessing before receiving a message to transmit. In that paper, the lower bound is proved using a layered structure, where a crucial assumption is that, in each layer, all nodes run the same uniform protocol upon receiving the message to be broadcasted. By the definition of a broadcast protocol given in that paper "any other processor is inactive until receiving a message for the first time". If preprocessing is allowed the protocol may be non-uniform as in our protocol.

Roadmap. In the remainder of this paper we define the models used througout in Section 2 and we present the details of our algorithm in Section 3.

2 The Model

Radio Networks is a vast area and there is a myriad of applications of such a technology. Depending on the application, topologies and node constraints may be very different. In Sensor Networks, nodes are expected to be deployed at random in large quantities over an area of interest and two nodes can communicate only if they are mutually in range. Thus, we model the topology as a *Geometric Graph*, where nodes are distributed arbitrarily in $\mathbb{R}^2$, and a pair of nodes is connected by an edge if and only if they are at an Euclidean distance of at most a parameter r. We also assume that the topology is unknown to the nodes and the only knowledge each node has is the number of nodes in the whole network n, its unique identifier in $\{1, \ldots, n\}$, and a constant parameter β to be defined later.

In addition to topology and connectivity models, an appropriate model of the constraints of the nodes in the network has to be defined in order to properly design and analyze protocols. Bar-Yehuda, Goldreich and Itai [1] used a formal model of a radio network that specifies many of those restrictions, including limits on contention resolution, but they make no mention of computational limits such as small memory. We use a relaxed version of the comprehensive Weak Sensor Model [10] where memory size is bounded only by $O(nm)$ where m is the message size. Briefly, the following assumptions are included in this model. The communication among neighboring nodes is through broadcast on a shared channel, where a node receives a message only if exactly one of its neighbors transmits. If more than one message is sent at the same time, a collision occurs and no collision detection mechanism is available. Sensors nodes cannot receive and transmit in the same time slot. The channel is assumed to have only two states: transmission and silence/collision. Time is assumed to be slotted and all nodes have the same clock frequency, but no global synchronizing mechanism is available. Furthermore, they *wake-up* adversarially. We assume that sensor

nodes can adjust their power of transmission but only to a constant number of levels. Other limitations include: limited life cycle, short transmission range, only one channel of communication, no position information, and unreliability.

3 An Optimal Algorithm

We describe now the gossiping protocol for Sensor Networks. Although global synchronism is not required, for the sake of clarity, we assume first that nodes start simultaneously and analyze each phase separately. Later we show that such an assumption is not necessary. The protocol has the following four main phases.

1. Define nodes as *master nodes* in such a way that every node is within distance at most ar of some master, and all master nodes are separated by a distance at least ar, where r is the maximum range of transmission and a is a constant such that $0 < a < 1/3$. All non-master nodes are called *slave nodes*. Notice that any node can be the slave of at most 6 masters.
2. Every master node reserves blocks of time steps for local use, so that each master and its slaves can communicate without colliding with transmissions from any nodes within radius r.
3. Every master node maintains a set of messages received, initially containing only its own message. Using the reserved blocks, all slave nodes transmit their message to their master nodes transmitting with radius ar. Every master node adds messages received from its slaves to its set.
4. Using the reserved blocks, every master node deterministically transmits its set of messages to all master nodes within radius at most r and repeatedly adds the messages received from other masters and re-transmits.

The choice of the upper bound on a guarantees that communication between master and slave nodes is achieved in a time slot that is not used by any neighboring master-slave pair. Given that a is a constant, its effect is folded in the other constants of the analysis. More precisely, as we will see in Section 3.2, the more master nodes that are included in a circle of radius r the bigger is the block of reserved time slots. Although the size of such a block is still a constant, for constant-sensitive applications a must be made as big as possible. We now give the details of each phase and the analysis.

3.1 Phase 1

This phase of the protocol can be implemented distributedly running a Maximal Independent Set (MIS) algorithm with radius ar. For that purpose we use the algorithm presented in [19] which works in two stages. In an initial bounding stage, the number of neighboring nodes that will participate in the second stage is upper bounded to $O(\log n)$. In a second stage, nodes keep a counter of the time passed since their first transmission or the last reception of a sufficiently close neighbor-counter. A long enough time without receiving a neighbor's counter enables a node to declare itself a member of the MIS with low probability of

error. The second stage, tailored for the Sensor Network setting was presented in [10]. We omit the details here for the sake of brevity.

Lemma 1. *Under the restrictions of the Weak Sensor Model, for a given node running the algorithm described above, at least one node within its transmission range joins the MIS in* $O(\log^2 n)$ *time steps and no two MIS nodes are within range of each other with probability* $1 - 1/n^{\gamma_1}$ *for some constant* $\gamma_1 > 0$.

Proof. As in [19] and [10]. □

3.2 Phase 2

We implement this phase using a counter to break symmetry as in the previous algorithm. The main idea is for each master node to reserve certain steps for deterministic transmissions in a way that there are no collisions.

Algorithm 1. Algorithm of Phase 2. α_1, α_2, α_3, α_4, α_5, b and β are constants.

```
1  for each master node do
2      set a step counter to 0.
3      while true do
4          if current step was not reserved then
5              transmit the counter and ID with probability 1/α1 and radius r
               using non-reserved slots.
6              if not transmitting in the current time slot then
7                  if a neighbor's counter is received and the absolute difference
                   between the local and neighbor's counter is ≤ α2 log n then
8                      set local counter to 0.
9                  else
10                     if a neighbor's reservation message is received then
11                         keep track of slots reserved.
12             increase counter if transmitted at least once.
13             if the counter reached ⌈α3 log n⌉ then
14                 choose a block of b contiguous available time slots in an interval
                   of β.
15                 for α5 log n available steps do
16                     transmit ID and the incoming time slots reserved with
                       probability 1/α4 and radius r using non-reserved slots.
17                 while true do
18                     transmit a beacon message in the reserved slot with radius
                       ar.
```

The protocol, detailed in Algorithm 1, works as follows. α_1, α_2, α_3, α_4, α_5, b and β are constants. Each master node x maintains a step counter, initially set to 0. In each step still not reserved by any of the master nodes within distance

r, x transmits its counter and its identity with probability $1/\alpha_1$ within a radius of r. In each step that x does not transmit, it is in receiver mode. If x receives the value of a neighbor's counter which is ahead or behind x's counter by less than $\alpha_2 \log n$, x resets its counter to 0. Upon reaching a final count of $\alpha_3 \log n$, x chooses a block of b contiguous available time slots to be used periodically with period β.

Next, x informs to the neighboring master nodes which are the slots chosen. In order to do that, x transmits a message containing the number of steps after the current step in which its reserved block takes place. This message is repeatedly transmitted with probability $1/\alpha_4$, radius r and using only non-reserved slots. Of course, the number of steps is updated appropriately in each step. As in [10], master nodes within distance of r from this node are guaranteed to receive this message within $O(\log n)$ steps and no neighboring master node can reach its final count before that w.h.p.

After $\alpha_5 \log n$ steps, the master node synchronizes its slaves by repeatedly transmitting a *beacon message*. After the first beacon message slave nodes move to phase 3.

The block of b reserved slots is big enough to include one for slave transmissions, one for master acknowledgements to slave transmissions, one for beacon messages, and one for transmissions among master nodes in the last phase. The period β is a constant big enough to ensure that each master node gets to reserve some block. As we show in Lemma 2, the number of master nodes in any circle of radius r is bounded by $O(1)$. Thus, such a constant value β exists.

Lemma 2. *There are at most* $3\lceil 2/a\sqrt{3}\rceil(\lceil 2/a\sqrt{3}\rceil + 1)$ *master nodes within distance* r *of any master node with high probability.*

Proof. All master nodes are separated by a distance of at least ar with high probability as a result of phase 1. Consider the smallest regular hexagon whose side is a multiple of ar and covers completely a circle of radius r. Consider a tiling of such hexagon with equilateral triangles of side ar. As proved by Fejes-Tóth in 1940 [11], the hexagonal lattice is indeed the densest of all possible plane packings. Therefore, the number of vertices in such a tiling minus one is an upper bound on the number of master nodes at a distance r of a master node located in the center of such a hexagon. That number is $3\lceil 2/a\sqrt{3}\rceil(\lceil 2/a\sqrt{3}\rceil + 1)$. □

Lemma 3. *After* $O(\log n)$ *time steps running the algorithm described above, any master node reserves a block of* $b \in O(1)$ *steps every* $\beta \in O(1)$ *steps for local use, i.e., this block does not overlap with the block of any other master node separated by a distance at most* r*, with probability* $1 - 1/n^{\gamma_2}$ *for some constant* $\gamma_2 > 0$.

Proof. The running time of the algorithm can be proved as in [10].

To complete the proof we consider two cases.

Case 1: we assume for the sake of contradiction that the blocks reserved by some pair of master nodes separated by a distance at most r overlap. This implies that at least one of them did not receive the message of the other. But, using the

techniques in [10] it can be shown that the probability of that event is $O(1/n^{\gamma_3})$ for some constant $\gamma_3 > 2$, which ensures that the probability of failure over all possible pairs is low.

Case 2: we assume for the sake of contradiction that a master node x can not reserve a contiguous block of b slots. This implies that after some neighboring master nodes reserve their blocks, there are no contiguous b slots available. As proved in Lemma 2 there are at most $3\lceil 2/a\sqrt{3}\rceil(\lceil 2/a\sqrt{3}\rceil + 1)$ master nodes within r distance of any master node w.h.p. But, making $\beta \geq (2b-1)(1 + 3\lceil 2/a\sqrt{3}\rceil(\lceil 2/a\sqrt{3}\rceil + 1))$ there is always a block of b contiguous slots available w.h.p. □

3.3 Phase 3

In this phase we need to guarantee that all slave nodes transmit their message to their master. A simple randomized algorithm can achieve this task in $O(\Delta \log n)$ but we show in this section that it can be done faster using the synchronism achieved in the previous phase.

In order to implement this phase efficiently, we could use an approach similar to the algorithm presented in [13]. This algorithm solves the problem of realizing arbitrary h-relations in an n-node network with high probability in $\Theta(h + \log n \log\log n)$ steps. In an h-relation, each processor is the source as well as the destination of h messages. However, the protocol requires that nodes know h, in our problem Δ. So, instead, we use an scheme where the only topology information is the size of the whole network n.

As explained before, slave nodes periodically receive a beacon message from their master node indicating the forecoming available slots for local use. A block of reserved slots includes, among others, a slot for slave transmissions and a slot for master acknowledgement. This acknowledgement informs a node that its transmission was successful, implementing a collision detection mechanism. Thus, we can take advantage of local synchronism achieved by the beacon message and collision detection implemented by the acknowledgement. For the sake of clarity, we focus here in the description of the algorithm ignoring these details and the fact that nodes use only the reserved slots for transmissions.

The protocol, detailed in Algorithm 2, works as follows. The algorithm is window-based, i.e., nodes repeatedly choose uniformly one time slot within an interval, or *window*, of time slots to transmit its message. Regarding the size of such a window, the protocol follows a back-on/back-off strategy, i.e., the window is increased in an outer loop by the master and decreased in an inner loop by the slaves. The master informs the slaves of the current window size in the beacon message. In order to succeed with high probability when $o(\log n)$ messages are left, $\Theta(\log^2 n)$ steps where nodes repeatedly transmit with probability $1/\log n$ are included at the end of each phase of the outer loop.

The intuition for the algorithm is as follows. Assume nodes know the number of nodes in their one-hop neighborhood, call it δ. Then, we think of the problem as a random process where δ balls (modelling the messages) are dropped uniformly in δ bins (modelling time slots). We will show that, for large enough δ,

Algorithm 2. Algorithm of Phase 3. α is a positive constant

```
1 for i = {⌊log log n⌋, ⌊log log n⌋ + 1, ⌊log log n⌋ + 2, ...} do
2     Each master node transmits i in the beacon message.
3     Each slave node does the following.
4     for j = 0 to i − 1 do
5         Choose uniformly a step within the next 2^(i−j) steps.
6         Transmit the message in such a step and receive messages in all the
          other steps.
7     for α log^2 n steps do
8         Transmit the message with probability 1/ log n.
9         Receive messages if not transmitting.
```

with high probability a constant fraction of the balls fall alone in a bin. Now, we can repeat the process removing this constant fraction of balls and bins until all balls have fallen alone. Since nodes do not know the size of their neighborhood, the outer loop increasing the number of bins is necessary.

We now concentrate in the analysis of this phase. First, we need the following lemma about bins and balls.

Lemma 4. *For $\delta \geq (2e/(1-e\epsilon)^2)(1+(\gamma_4+1/2)\ln n)$, if δ balls are dropped in δ bins uniformly at random, the probability that the number of bins with exactly one ball is smaller than $\epsilon\delta$ is at most $1/n^{\gamma_4}$ for some constants $\gamma_4 > 0$ and $0 < \epsilon < 1$.*

Proof. The probability for a given ball to fall in a given bin is $(1/\delta)(1-1/\delta)^{\delta-1} \geq 1/e\delta$. Let X_i be a random variable that indicates if there is exactly one ball in bin i. Then, $Pr(X_i = 1) \geq 1/e$. To handle the dependencies that arise in balls and bins problems, we approximate the joint distribution of the number of balls in all bins by assuming the load in each bin is an independent Poisson random variable with mean 1. Let X be a random variable that indicates the total number of bins with exactly one ball. Then, $\mu = E[X] = \delta/e$. Using Chernoff-Hoeffding bounds,

$$Pr(X \leq \epsilon\delta) \leq exp\left(-\frac{\delta}{2e}(1-e\epsilon)^2\right).$$

As shown in [18], any event that takes place with probability p in the Poisson case takes place with probability at most $pe\sqrt{\delta}$ in the exact case. Then, we want to show

$$exp\left(-\frac{\delta}{2e}(1-e\epsilon)^2\right) e\sqrt{\delta} \leq n^{-\gamma}.$$

Which is true for

$$\delta \geq \frac{2e}{(1-e\epsilon)^2}\left(1+\left(\frac{1}{2}+\gamma\right)\ln n\right).$$

□

Lemma 5. *The algorithm described above guarantees that a master node receives the message of all its slaves within* $O(\Delta + \log^2 n \log \Delta)$ *steps with probability* $1 - 1/n^{\gamma_5}$ *for some constant* $\gamma_5 > 0$.

Proof. If $\delta \in O(\log n)$, as shown in [10], all slave nodes achieve a non-colliding transmission within $O(\log^2 n)$ steps with probability at least $1 - n^{-\gamma_i}$ for some constant $\gamma_i > 0$.

If $\delta \in \omega(\log n)$, after the master node transmits a window size in $\Theta(\delta)$, a constant fraction ϵ of slave nodes succeed in each step j of the inner loop with probability $1 - n^{-\gamma_j}$ as shown in Lemma 4.

Taking each γ_j small enough and ϵ big enough and telescoping the running time of each loop, the claim follows. □

3.4 Phase 4

Each master node maintains a set of messages, initially containing only its own message, adding all messages received in phases 3 and 4, and deterministically re-transmiting this set in the time slots reserved for that purpose.

Lemma 6. *Any master node running the algorithm of phase 4 receives all messages from other master nodes within* $O(D)$ *time steps, where* D *is the diameter of the network.*

Proof. Given that the master nodes form a maximal independent set, the diameter of the subgraph induced by them is in $O(D)$. Since master nodes re-transmit all messages ever received deterministically every $\beta \in O(1)$ steps, the claim follows. □

3.5 Overall Analysis

Two important restrictions of the Weak Sensor Model are that nodes start running the algorithm, or wake-up for short, according to an adversarial schedule, and that their power supply is unrealiable resulting in potential on/off cycles. In this section, we remove the assumption of simultaneous wake-up used in the analysis and we show that in fact the algorithm and its efficiency are still the same. Given that in order to solve the gossiping problem all nodes have to be active, we analyze the time after the last node starts running the algorithm and we assume that no node turns off before completion. Otherwise, any time analysis would be meaningless in presence of an adversary that turns on and off nodes forever.

The MIS algorithm used in phase 1 includes an initial waiting period, long enough to ensure that nodes waking-up do not interfere with nodes already running the algorithm. We extend this waiting period to the duration of the first two phases of the protocol. If during the waiting period a node becomes a slave of some master node, the slave waits for the beacon message doing nothing and goes directly to the third phase after receiving it. If a node does not become

a slave during its waiting period, it starts phase 1 using slots still available, i.e., slots that were not reserved by some master node within radius r. Choosing $\beta \in O(1)$ big enough, there is always some slot available every β slots. Choosing the constant factors of the probabilities appropriately, nodes running phases 1 and 2 do not interfere with each other w.h.p. as proved in [10].

Nodes in phase 3 are synchronized by their master beacon-message. However, if a node is woken up late enough with respect to its slave neighbors, it could reach this phase after the window size of the outer loop is in $\omega(\Delta)$. In order to avoid this situation, whenever the master node does not receive transmissions after receiving transmissions for a given window size, it resets the outer loop. Given that the running time is analyzed after the last node is woken up the claimed running time still holds.

Given that nodes running phases 1 and 2 use all slots not reserved there is also no conflict with nodes running phases 3 and 4. The last two phases are deterministic and utilize time multiplexing, synchronized by the beacon message. Thus, there is also no conflict among nodes in these phases.

A straightforward application of the lemmata of previous sections, gives our main theorem.

Theorem 1. *Given a network of n nodes, after the last node starts running the algorithm described in this section, the gossiping problem is solved with high probability in $O(D + \Delta)$.*

Proof. Using Lemmas 1, 3, 5, and 6 the overall complexity of the algorithm including preprocessing is $O(\log^2 n + \log n + \Delta + \log^2 n \log \Delta + D)$ with high probability. Given the geometric constraints, the number of one-hop neighborhoods is bounded by $O(D^2)$. In addition, the maximum number of nodes in any one-hop neighborhood is at most Δ. Hence, D and Δ can not be simultanoeusly in $o(n^c)$ for any constant $c > 0$ and the claim follows. □

Given that $\Omega(D)$ and $\Omega(\Delta)$ are lower bounds for this problem, the previous result is tight.

References

1. Bar-Yehuda, R., Goldreich, O., Itai, A.: On the time-complexity of broadcast in multi-hop radio networks: An exponential gap between determinism and randomization. Journal of Computer and System Sciences 45, 104–126 (1992)
2. Bar-Yehuda, R., Israeli, A., Itai, A.: Multiple communication in multihop radio networks. SIAM Journal on Computing 22, 875–887 (1993)
3. Bruschi, D., Pinto, M.D.: Lower bounds for the broadcast problem in mobile radio networks. Distributed Computing 10(3), 129–135 (1997)
4. Chlamtac, I., Weinstein, O.: The wave expansion approach to broadcasting in multihop networks. In: Proc. of INFOCOM (1987)
5. Chlebus, B., Gąsieniec, L., Lingas, A., Pagourtzis, A.: Oblivious gossiping in ad-hoc radio networks. In: Proc. of 5th International Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications, pp. 44–51 (2001)

6. Chrobak, M., Gąsieniec, L., Rytter, W.: Fast broadcasting and gossiping in radio networks. In: Proc. of the 41st IEEE Annual Symp. on Foundation of Computer Science (2000)
7. Chrobak, M., Gąsieniec, L., Rytter, W.: A randomized algorithm for gossiping in radio networks. Networks 43(2), 119–124 (2004)
8. Clementi, A., Monti, A., Silvestri, R.: Selective families, superimposed codes, and broadcasting on unknown radio networks. In: Proc. of the 12th Annual ACM-SIAM Symposium on Discrete Algorithms (2001)
9. Czumaj, A., Rytter, W.: Broadcasting algorithms in radio networks with unknown topology. In: Proc. of the 44th IEEE Annual Symp. on Foundation of Computer Science (2003)
10. Farach-Colton, M., Fernandes, R.J., Mosteiro, M.A.: Bootstrapping a hop-optimal network in the weak sensor model. In: Brodal, G.S., Leonardi, S. (eds.) ESA 2005. LNCS, vol. 3669, pp. 827–838. Springer, Heidelberg (2005)
11. Fejes-Tóth, L.: über einen geometrischen satz. Mathematische Zeitschrift 46(1), 83–85 (1940)
12. Gąsieniec, L., Potapov, I.: Gossiping with unit messages in known radio networks. In: Proc. of 17th IFIP World Computer Congress / 2nd IFIP International Conference on Theoretical Computer Science (2002)
13. Geréb-Graus, M., Tsantiles, T.: Efficient optical communication in parallel computers. In: 4th Annual ACM Symposium on Parallel Algorithms and Architectures pp. 41–48 (1992)
14. Kowalski, D.R.: On selection problem in radio networks. In: Proceedings 24th Annual ACM Symposium on Principles of Distributed Computing, pp. 158–166 (2005)
15. Kowalski, D.R., Pelc, A.: Time of deterministic broadcasting in radio networks with local knowledge. SIAM Journal on Computing 33(4), 870–891 (2004)
16. Kushilevitz, E., Mansour, Y.: An $\Omega(D \log(N/D))$ lower bound for broadcast in radio networks. SIAM Journal on Computing 27(3), 702–712 (1998)
17. Liu, D., Prabhakaran, M.: On randomized broadcasting and gossiping in radio networks. In: Ibarra, O.H., Zhang, L. (eds.) COCOON 2002. LNCS, vol. 2387, pp. 340–349. Springer, Heidelberg (2002)
18. Mitzenmacher, M., Upfal, E.: Probability and Computing: Randomized Algorithms and Probabilistic Analysis. Cambridge University Press, Cambridge (2005)
19. Moscibroda, T., Wattenhofer, R.: Maximal independent sets in radio networks. Technical Report TR-478, Computer Engineering and Networks Laboratory, ETH Zurich, 8092 Zurich, Switzerland (December 2004)
20. Ravelomanana, V.: Optimal initialization and gossiping algorithms for random radio networks. IEEE Trans. Parallel Distr. Syst. 18(1), 17–28 (2007)

On the Complexity of the "Most General" Undirected Firing Squad Synchronization Problem

Darin Goldstein and Kojiro Kobayashi

[1] The first author received private funding for this research
Academic information: Department of Computer Engineering and Computer Science
California State University, Long Beach
dgoldste@csulb.edu
[2] Department of Information Systems Science
Soka University
kobayasi@t.soka.ac.jp

Abstract. We show that if a minimal-time solution exists for a fundamental distributed computation primitive, synchronizing arbitrary undirected networks of finite-state processors, then there must exist an "extraordinarily fast" $\tilde{O}(D^5E)$ algorithm[1] in the RAM model of computation for exactly determining the diameter D of an arbitrary unweighted undirected graph with E edges. The proof is constructive.

At present we know eight variations of the firing squad synchronization problems whose solutions are known but whose minimal-time solutions are not known. Our result essentially completes the program outlined in [3] to show that it is highly unlikely for there to exist minimal-time solutions for these variations.

1 Introduction

The firing squad synchronization problem (FSSP) is a famous problem originally posed almost half a century ago with many variations. *FSSP for undirected networks*, or UN(k) for short (k (≥ 3) is a parameter of the problem), is one of its variations and can be formulated as follows.

The problem is to design a finite-state machine A that satisfies several conditions. The machine A has k terminals. Each of the terminals works as both an input terminal and output terminal. We construct a network N from copies of A by connecting their terminals with bidirectional connections. The value of i-th terminal as an output terminal at a time t is the pair (i, s) of its terminal number i and the state s of the machine at the time. As an input terminal, the value of a terminal is the value of the output terminal with which it is connected. A terminal may be open and in that case it received a special value # as an

[1] We use the notation $\tilde{O}$ to represent standard asymptotic notation that ignores logarithmic terms in the variables. Thus, for example, $D \log E = \tilde{O}(D)$.

T. Tokuyama (Ed.): ISAAC 2007, LNCS 4835, pp. 244–255, 2007.

input terminal. The state of a machine at time $t+1$ is determined by its state and the values of its k input terminals at time t.

The machine A has at least three different states Q (the quiescent state), G (the "general" state), and F (the firing state). Intuitively, Q is an inactive state. If the state of A is Q and all values of its input terminals are either of the form (i, Q) or # at a time t, its state at time $t+1$ must be Q. One of the copies of a network N is specified as the "general" of the network. Intuitively this is the unique copy that is activated at time 0. At time 0 the state of a copy in a network N is G if it is the general and Q otherwise ("soldiers"). Intuitively F is the state of the machine in which it performs some action ("fire").

The goal of the problem UN(k) is to design a machine A so that, for any network N of copies of A there exists a time t_0 such that the state of all copies of N are F at the time t_0 and the state of any copy at any time before the time t_0 is not F. In other words, all the copies of the networks N should enter the firing state F simultaneously for the first time.

We call a machine A that satisfies these conditions a *solution* of UN(k). We call the time t_0 the *firing time* of the solution A for the network N. Assuming that there exists a solution, for each network N we define the *minimum firing time* of N as the minimum of firing times of solutions A for N, where the minimum is over all solutions A. Moreover, we call a solution $\tilde{A}$ a *minimal-time solution* if its firing time for N is the minimum firing time of N for any network N.

The problem itself is interesting as a mathematical puzzle. More importantly, there are also applications to the synchronization of small, fast processors in large networks. In the literature on the subject, the problem has been referred to as "macrosynchronization given microsynchronization" and "realizing global synchronization using only local information exchange." The synchronization of multiple small but fast processors in general networks is a fundamental problem of parallel processing and a computing primitive of distributed computation.

2 Variations of FSSP

The original FSSP was the one in which networks are restricted to linear arrays and the general is the leftmost copy. There are many variations of the original FSSP, for example, linear arrays with arbitrary position of the general, rings, rectangles, cubes, directed networks, and undirected networks (UN(k)). To save space, we omit the history and overview of these variations and refer the reader to [4] and [1].

For many of these variations, minimal-time solutions are known. However, there are several variations in which solutions are known but minimal-time solutions are not known. In the following we will mention eight such variations. At present, for each of them there exists no proof of nonexistence of minimal-time solutions.

In the first variation 2PATH, a network is a path in the two-dimensional grid space and the general is at one of the two endpoints. The second variation g-2PATH is similar to 2PATH but the general may be at any position ("g" is for

"*g*eneralized"). In the third variation 2REG, a network is a connected region in the two-dimensional grid space and the general may be at any position. Three other variations 3PATH, g-3PATH, 3REG are similar variations for the three-dimensional grid space. Finally, DN(k) is the variation for directed networks ($k \geq 2$), and UN(k) is the variation mentioned previously (the variation for undirected networks). In DN(k), a finite-state machine A has k input terminals and k output terminals and connections are unilateral.

The variation 2PATH was originally studied by Kobayashi in [3]. The two-dimensional path extension problem, or 2PEP for short, is the problem to decide, for each given self-avoiding path in the two-dimensional grid space, whether there exists a valid extension of the path such that the length is doubled. In [3], it is shown that if 2PATH has a minimal-time solution, then 2PEP is solvable in time $O(n^2)$ on a Turing Machine. Though 2PEP seems like a difficult problem, its time complexity has yet to be resolved. This result also readily applies to the other two-dimensional cases, g-2PATH and 2REG.

The remainder of the problems were studied by Goldstein and Kobayashi [1,2]. We could show that, for each of the three-dimensional variations 3PATH, g-3PATH, and 3REG, if $P \neq NP$ then the problem has no minimal-time solution.

In [2], the authors showed that, for any k (≥ 2), if DN(k) had a minimal-time solution, then there must exist an "extraordinarily fast" $\tilde{O}((E+nD)D)$ algorithm in the RAM model of computation for exactly determining the diameter of an arbitrary unweighted directed graph.

Therefore, for seven of the eight variations 2PATH, g-2PATH, 2REG, 3PATH, g-3PATH, 3REG, DN(k), UN(k), we know that finding a minimal-time solution is at least as difficult as designing a very efficient algorithm for some problem.

In this work, we show that if a minimal-time solution exists for UN(k), then there must exist a $\tilde{O}(D^5E)$ algorithm for exactly determining the diameter of an arbitrary unweighted *undirected* graph. This will essentially complete the program initiated in [3] to show that, for the eight variations mentioned above it is highly unlikely for there to exist minimal-time solutions.

This result is curious. Note the difference between the $\tilde{O}((E + nD)D)$ algorithm presented in [2] and the $\tilde{O}(D^5E)$ algorithm presented here. Why should the arbitrary *undirected* topology require more effort than the arbitrary *directed* topology? One might believe upon seeing the proof of the directed case that the undirected case would be a simple straightforward adaptation. In fact, the necessary proof for the undirected case is significantly more complex and intricate than the proof for the directed case, and the reasons seem to be inherent to the problem itself (though, of course, we do not claim to have a proof of this).

3 The Results

3.1 Initial Definitions, Lemmas, and Constructions

Henceforward, we will identify a network N of copies of a finite-state machine A with the underlying graph G that represents the connections. We also call the general of a network its *root*.

Definition 1. *Given a* directed *graph G, define $I(v)$ to be the set of edges with terminal vertex v. For $e \in I(v')$, define $d_e(v, v')$ to be the length of the shortest path from v to v' such that the final edge in the path is e. Then for two vertices $v, v' \in G$, define $d^*(v, v')$ to be* $\max_{e' \in I(v')} d_{e'}(v, v')$.

To calculate the value of $d^*(v, v')$ in an undirected *graph, we replace each undirected edge in G with two unidirectional edges, one in each direction. We now specify that the value of $d^*(v, v')$ in the undirected case is equal to the value computed when the bidirectional edges are replaced with two unidirectional edges.*

Lemma 1. *For each k (≥ 1) and each problem instance $G = (V, E)$ of UN(k) that is not the singleton network (the network with one node and no connections), we have*

$$mft(G) = \max_{v,v' \in V} \{d^*(v_{root}, v) + d(v, v')\}$$

for the minimum firing time $mft(G)$ of G.

Lemma 2. *For each $k \geq 3$, if UN(k) has a minimal-time solution, then UN(3) has a minimal-time solution.*

The remainder of this section will be devoted to proving the following theorem.

Theorem 1. *If there exists a minimal-time solution for FSSP on the general undirected network topology,* UN(k)*, for any $k \geq 3$, then there exists a deterministic algorithm in the RAM model of computation that can exactly determine the diameter of a general unweighted directed graph in time $\tilde{O}(D^5 E)$ where E is the number of edges and D is the diameter of the graph,*

We will assume that we are given an arbitrary undirected graph $G = (V, E)$ with n nodes and are operating in the RAM model of computation. A simple depth-first search can be used to determine in time $O(V + E)$ time whether the graph has infinite diameter or not. We will assume that the diameter is found to be finite. We can also check in $O(E)$ time whether or not the graph has diameter 1; we will assume that the graph G has diameter strictly greater than 1.

Note that via Lemma 2, it will suffice to prove the result for $k = 3$. Thus, for the remainder of the proof, we will assume $k = 3$.

We now proceed with a series of transformations of the graph G.

Transforming G to $G'(s)$. Our main goal in this section will be to transform the graph G of potentially unbounded degree to a graph $G'(s)$ with degree exactly 3. We perform this transformation in stages.

- Place a node in the center of every edge. In other words, replace each edge $v_1 \overset{e}{\leftrightarrow} v_2$ with $v_1 \overset{e_1}{\leftrightarrow} v' \overset{e_2}{\leftrightarrow} v_2$. This effectively doubles the length of each edge. Let this new graph be $G_1 = (V_1, E_1)$. We refer to the set of vertices that were originally in the graph G as V. (In this case, $v_1, v_2 \in V$ and $v_1, v', v_2 \in V_1$.)
- Vertices in G_1 may have potentially unbounded degree. Our goal is now to get the degree of the graph down to at most 3. We therefore replace each vertex $v \in G_1$ with a cascaded binary tree of depth $N = \lceil \log_2 n \rceil$ with root v. We connect

each edge incident on v to the leaves of this binary tree. Call this altered graph G_2, and note that G_2 has maximum degree 3. We refer to the set of vertices V_1 as the set of roots of these binary trees, vertices that were in the graph G_1. Note also that each $v \in V_1$ at this point only has degree 2.

– In G_2, let us consider the path from two vertices v_1 and v_2 that were adjacent in the graph G. Essentially, the path between the two vertices is as follows: start at v_1, exit through v_1's binary tree, pass across the edge $e_{v_1,v'}$, enter and exit through v''s binary tree, pass across edge e_{v',v_2}, enter v_2's binary tree, and finally stop at v_2. In this final transformation stage, we replace the edges that are not in any binary tree ($e_{v_1,v'}$ and e_{v',v_2} in this case) with a sequence of $2s$ edges, lengthening them by a factor of $2s$. *The value of s will be a known constant whose value we will specify later.* Let the center of this long string of vertices be modified as shown in Figure 1.

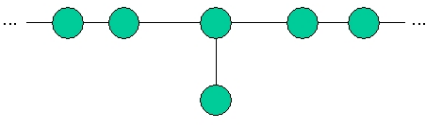


Fig. 1. This illustrates the modification to the center vertex of the long string of $2s$ vertices, transforming it into 2 vertices as shown

The result of performing this sequence of transformations will be the graph $G'(s)$.

Transforming $G'(s)$ to $G''(s, r)$. We now create a graph $G''(s, r)$ from $G'(s)$. First, create $|V_1|$ bidirectional paths of length r, for some number r to be determined later. These are the *connectors*. Connect one end of each of these connectors to the "extra" port in each vertex in G_1. Create a root node and have the root cascade outwards, each branch of length $\lceil \log_2 |V_1| \rceil$. Have these link up with the other end of the connectors.

In terms of notation, B and C will refer to the set of vertices in the lower binary tree and the connectors respectively.

Transforming $G''(s, r)$ to $G''(x, s, r)$. When we perform our calculations, we will need an additional graph similar to $G''(s, r)$ but with some slight modifications. To form $G''(x, s, r)$ from $G''(s, r)$, we perform the following operations.

– For each connector that attaches to a vertex in G, elongate the connector by x units.
– Elongate each path between two vertices in G_1 by x units, and move each of the "hangers" (illustrated in Figure 1) towards the vertices in G so that the distance from the hanger to the nearest vertex in G is unchanged. Thus, the total distance between two adjacent vertices in V_1 is $2N + 2s + x$ after this modification.

Note that we will not use $G''(x,s,r)$ explicitly until the very end of the algorithm, but we need to prove some facts about it. It is important to note that $G''(0,s,r) = G''(s,r)$.

Also, note that this transformation implicitly transform the graph $G'(s)$ into a graph $G'(x,s)$. $V'(x,s)$ will refer to the vertices within $G'(x,s)$.

3.2 Calculations

In this section, we will perform some of the calculations needed for the algorithm.

Definition 2. *For a graph G, let the notation D_G represent the diameter of the graph G.*

Definition 3. *Let*

$$f(v,v') = d^*(v_{root},v) + d(v,v')$$

Recall that by Lemma 1, the minimal firing time of any network G is $\max_{v,v' \in G} f(v,v')$.

Before proceeding to the analysis, we need a few preliminary lemmas.

Preliminary Lemmas

Lemma 3. *Assume that $v,v' \in V''(x,s,r)$ maximize the value of the function f. Then we can assume without loss of generality that $v \in V'(x,s)$ and that v is a vertex that is distance $N+s$ from a vertex in G. (This places v just above some hanger.)*

Definition 4. *Let $V_v \subset V'(x,s)$ be the set of vertices that are a distance $N+s$ from a vertex in G. We choose these names because if $v,v' \in V''(x,s,r)$ maximize f, then by Lemma 3, without loss of generality $v \in V_v$.*

Definition 5. *Let the quantity $d_r(v,v')$ (resp. $d_n(v,v')$) be the length of the shortest path from v to v' that does (resp. does not) pass through the lower binary tree. If v' happens to be in the lower binary tree, then we define $d_r(v,v') = d_n(v,v') = d(v,v')$.*

Assume that v,v' maximize f in $G''(x,s,r)$. We know that we can assume $v \in V_v$ by Lemma 3, but we have three possible locations for v': B, C, and $V'(x,s)$. We will examine the three possibilities for the position of the vertex v' and evaluate the value of the function f in each case. To simplify our calculations, we note that because $v \in V_v$, the value of $d^*(v_{\text{root}},v) = \log_2|V_1| + r + N + x + s + 2$ is fixed and does not depend on the position of v'. Thus, when maximizing the quantity f, we only need to consider the quantity $\max_{v \in V_v, v' \in V''(x,s,r)} d(v,v')$.

Lemma 4. $N \leq \log_2|V_1| \leq 2N$ *if* $n > 1$.

Below, we will make repeated use of the following fact: If $s > 2N$ (which we will be assuming is always true in the algorithm), then because $2s > 4N \geq 2\log_2|V_1|$

(by Lemma 4), any shortest path from $v \in V_v$ to $v' \in C$ that passes through the lower binary tree will, once having left $V'(x,s)-C$, never again enter $V'(x,s)-C$. Thus, if $v \in V_v$ and $v' \in C$, then the value of $d_r(v,v')$ is higher the closer v' gets to $V'(x,s)$ and lower the further v' gets from $V'(x,s)$.

Definition 6. – *Let $g : C \to G_1$ be the map that takes a vertex $v \in C$ and maps it to the unique vertex in G_1 above it in the connector.*
– *Let $h : C \to B$ be the map that takes a vertex $v \in C$ and maps it to the unique vertex in B directly below it in the connector.*

Lemma 5. *If $r \geq s \geq 6N$, $\max_{v \in V_v, v' \in C \cup V'(x,s)} d(v,v') > s + r + 5N + x$.*

Corollary 1. *If $r \geq s \geq 6N$,*

$$\max_{v,v' \in V''(x,s,r)} d(v,v') = \max_{v \in V_v, v' \in C \cup V'(x,s)} d(v,v')$$

Definition 7. *Define $C^* \equiv C - V'(x,s) - B$.*

Lemma 6. *Assume that v, v' maximize f in $G''(x,s,r)$ and $r \geq D_{G'(x,s)}$. Then $v' \in C^*$.*

Lemma 7. *Let v, v' maximize f within $G''(x,s,r)$. Assume that $v' \in C^*$ and $s > 2N$. Then we can conclude*

1. $|d_r(v,v') - d_n(v,v')| \leq 1$
2. $d(v,v') = \lfloor \frac{d_r(v,v') + d_n(v,v')}{2} \rfloor$

Lemma 8. *Assume that $v \in V_v \subset V''(x,s,r)$, $r \geq s > 4N$, and let $v' \in C$. Then*

$$d_r(v,v') + d_n(v,v') = d_r(v, g(v')) + d_n(v, g(v'))$$

Definition 8. *Fix any vertex $v \in V'(x,s)$. We denote by v^V, the vertex $w \in V$ such that $d_n(v,w)$ is minimized. A similar definition holds for $v^{V_1 - V}$. Note that for any vertex $v \in V_v$, the vertices v^V and $v^{V_1 - V}$ by construction appear on opposite ends of the edge in $V'(x,s)$ on which v appears.*

Lemma 9. *Let $v \in V_v \subset V''(x,s,r)$ and let $v' \in C$. Consider a shortest path from v to v' that does not pass through the lower binary tree. Let v^* be a vertex along this path such that $d_n(v^*, v') \leq s - N$ and if v^* is not on the same connector as v' then $v^* \in V'(x,s)$. Then either v^* is on the same connector as v' or the path realized by $d_r(v, v^*)$ is required to pass through $g(v')$ and v'.*

Definition 9. *If $v \in V_v$ and $v' \in C$, define*

$$d_{G_1}(v,v') \equiv \min\{d_{G_1}(v^V, g(v')), d_{G_1}(v^{V_1 - V}, g(v'))\}$$

Corollary 2. *Let v, v' maximize f in $G''(x,s,r)$. By Lemma 3, we can assume without loss of generality that $v \in V_v$. Assume that $v' \in C^*$ and $s \geq N(d_{G_1}(v,v') + 3) + x$. Then $d_{G_1}(v,v') = D_{G_1} - 1$.*

Calculations within $G''(s,r)$. In this section, we will make calculations that deal with $G''(s,r) = G''(0,s,r)$. Lemmas in this section are specific to $G''(s,r)$; we ignore the x value so as to simplify notation.

Before proceeding to the following lemma, we make a quick observation. Note that if r is large enough and s is small compared to r, then if $v_1, v_2 \in V'(s)$, most values of $d_r(v_1, v_2)$ are relatively close to each other in comparison with r. In fact, we can put a quantitative bound on the maximum difference between two such values:

Lemma 10. *If $v_1, w_1 \in V_v$ and $v_2, w_2 \in V'(s)$, then*

$$d_r(v_1, v_2) < d_r(w_1, w_2) + 2s + 6N$$

Lemma 11. *Let $s > 4N$ and $r \geq 2s + 3N$. Assume that $v, v' \in V''(s,r)$ maximize f and that $v' \in V'(s)$. Consider the following two statements.*

1. $d_r(v,v') < d_n(v,v') + 2s + 6N$
2. $d_n(v,v') = d(v,v') = \max_{b \in V_v, b' \in V'(s)} d_n(b,b') \leq d_r(v,v') \leq d_n(v,v') + 4N + 4s$

If Statement 1 is not true, then Statement 2 must be true.

By Corollary 1 and Lemma 11, if $v, v' \in V''(s,r)$ maximize f, $s > 4N$, and $r \geq 2s + 3N$, then there are three (not necessarily mutually exclusive) regimes that v' may fall within. In all cases, by Lemma 4, without loss of generality $v \in V_v$.

1. $v' \in V'(s)$ and $d_r(v,v') < d_n(v,v') + 2s + 6N$
2. $v' \in V'(s)$ and $d_n(v,v') = d(v,v') = \max_{b \in V_v, b' \in V'(s)} d_n(b,b') \leq d_r(v,v') \leq d_n(v,v') + 4N + 4s$
3. $v' \in C^*$

Henceforward, whenever we refer to a graph as being in one of these three regimes, we are implicitly assuming that $s > 4N$ and $r \geq 2s + 3N$. (When we perform the algorithm below, these two restrictions will always be true.)

Regime Calculations for $G''(s,r)$. Our goal is now to show that if we predictably increase the value of r, for large enough r, the regime transitions are also somewhat predictable. Again, this section deals only with $G''(s,r) = G''(0,s,r)$.

Lemma 12. *The firing time of a regime 1 graph $G''(s,r)$ is greater than or equal to $3r - 3N + 4$. The firing time of a regime 1 graph $G''(s,r)$ is less than or equal to $3r + 3s + 9N + 3$.*

Lemma 13. *Assume that $G''(s,r)$ and $G''(s,4r)$ are both regime 2. Then the difference in firing times between the two graphs is exactly $3r$.*

Lemma 14. *Assume that $G''(s,r)$ and $G''(s,4r)$ are both regime 3 and $s \geq 6N$. Then the difference in firing times between the two graphs is at least $6r - 8N$ and at most $6r + 8N$.*

Lemma 15. *Assume that neither $G''(s,r)$ nor $G''(s,4r)$ are regime 1 graphs. If $G''(s,r)$ is a regime 3 graph, then $G''(s,4r)$ is also a regime 3 graph.*

The Estimation Lemma

Definition 10. *If $x < y$, we use the notation $[x,y]$ to indicate an unknown number between x and y.*

In the following lemma, the value $O(\frac{N}{s}d_{G_1}(v,v'))$ denotes a value X such that $X \leq c\frac{N}{s}d_{G_1}(v,v')$ for any G, s, and r. The value of the constant c is independent of G, s, and r.

Lemma 16. *Assume that $G''(s,r)$ is in regime 3. From the value of the minimal firing time of $G''(s,r)$, it is possible to estimate $d_{G_1}(v,v')$ to within $O(\frac{N}{s}d_{G_1}(v,v'))$.*

4 The Algorithm

We will describe the algorithm as a sequence of steps, proving their correctness as we go along. Let s start at $6N$ and let r start at $2s+3N$.

1. Our first order of business is to guarantee that we eventually produce an r value such that the graphs $G''(s,r)$ and $G''(s,4r)$ are both regime 3. Construct $G''(s,r)$ and $G''(s,4r)$ and note the firing times of both.
   - If the firing time of $G''(s,r)$ (resp. $G''(s,4r)$) is anywhere between $3r-3N+4$ and $3r+3s+9N+3$ (resp. $12r-3N+4$ and $12r+3s+9N+3$), we double the value of r and start again. If the firing time is outside of this range, then we know by Lemma 12 that the graphs $G''(s,r)$ and $G''(s,4r)$ cannot be in regime 1. We claim that regime 3 graphs must quickly move out of this range because the r dependence increase in the firing time has at least a factor of 6 by Lemma 14. Because all graphs become regime 3 for large enough r by Lemma 6 with $x=0$, the firing times of $G''(s,r)$ and $G''(s,4r)$ must eventually cease being in this range.
   - We also start again if the value of the difference in the two firing times is anything less than $6r-8N$ or more than $6r+8N$. The fact that the firing times must eventually lie within this range is again guaranteed by Lemma 6 with $x=0$ and Lemma 14.
2. At this point, both graphs are in regimes 2 or 3. By Lemma 13, both graphs cannot be in regime 2. Thus, at least one must be in regime 3. By Lemma 15, $G''(s,4r)$ must be in regime 3. Quadruple r so that $G''(s,r)$ is guaranteed to be regime 3.

   Throughout this section, we will use the following notation consistently: Let v, v' maximize f in $G''(s,r)$. By Lemma 3, $v \in V_v$, and because $G''(s,r)$ is now in regime 3, $v' \in C^*$.

   Using Lemma 16 and $G''(s,r)$, we can estimate the value of $d_{G_1}(v,v')$.

3. We continue doubling the value of s and starting again from the beginning (resetting r to equal twice the value of this new doubled s plus $3N$) until D_{G_1} has a unique value. In other words, we check the following two items in order. (Obviously, we check the first before we check the second. If the first fails, we automatically restart.)
   (a) The error estimation in the calculations of Lemma 16 for $d_{G_1}(v, v')$ yields a value strictly less than $\frac{1}{2}$. In other words, we know the exact value of $d_{G_1}(v, v')$.
   (b) The value of s satisfies $s \geq N(d_{G_1}(v, v') + 3)$. By Corollary 2, we have that $d_{G_1}(v, v') = D_{G_1} - 1$ and therefore we know the exact value of D_{G_1}.
4. Unfortunately, just because we know the value of D_{G_1} does not mean that we know the value of D_G. Consider the following cases.
   (a) There exists a diameter-length path in G_1 with two endpoints in V, and D_{G_1} is even. We can conclude that $D_G = \frac{D_{G_1}}{2}$.
   (b) Both endpoints of the diameter of G_1 are required to be in $V_1 - V$, and D_{G_1} is even. We can conclude that $D_G = \frac{D_{G_1}-2}{2}$ and v' lies in the connector of a vertex in $V_1 - V$.
   (c) D_{G_1} is odd. If D_{G_1} is odd, then we know that the maximal path has exactly one endpoint on a vertex of G. Thus, if D_{G_1} is odd, we can conclude that $D_G = \frac{D_{G_1}-1}{2}$.

   The remainder of the algorithm will be determining, if necessary, which of the former two possibilities actually occurs. From this point onwards, we will assume that D_{G_1} is even.
5. There are two cases that need to be considered concerning the structure of the graph G_1.
   (a) There exists two vertices $v_f, v_t \in V$ such that $d_{G_1}(v_f, v_t) = D_{G_1}$.
   (b) All diameter length paths in G_1 start and end at a vertex in $V_1 - V$.

   Let $x^* = N(D_{G_1} + 3), s^* = N(D_{G_1} + 3) + x^*, r^* = (D_{G_1} + 1)(x^* + 2s^* + 2N)$, and let $L = 2r^* + 2s^* D_{G_1} + x^* D_{G_1} + 4N + 2N D_{G_1}$. Form the graph $G''(x^*, s^*, r^*)$. Let w, w' maximize f in $G''(x^*, s^*, r^*)$. Note that by Lemma 6, $w' \in C^*$.

   Let us first assume that G_1 is a Case 5b graph. Then we must have $g(w') \in V_1 - V$. We can upper bound the value of $d_r(w, w') + d_n(w, w')$ as follows. Note that the first vertex in V_1 in a path realized by $d_n(w, w')$ must be in V because G_1 is bipartite.

$$
\begin{aligned}
&d_r(w, w') + d_n(w, w') \\
&= d_r(w, g(w')) + d_n(w, g(w')) \\
&\leq s^* + N + x^* + r^* + 2\log_2 |V_1| + r^* + d_n(w, g(w')) \\
&\leq s^* + x^* + 2r^* + 5N + d_n(w, g(w')) \\
&\leq s^* + x^* + 2r^* + 5N + s^* + (2N + 2s^* + x^*)(D_{G_1} - 1) + N \\
&= 2s^* + x^* + 2r^* + 6N + (2N + 2s^* + x^*)(D_{G_1} - 1) = L.
\end{aligned}
$$

   Now let us assume that G_1 is a Case 5a graph. Then $\exists v_f, v_t \in V$ such that $d_{G_1}(v_f, v_t) = D_{G_1}$. Let $v \in V_v$ be any vertex in an edge adjacent to v_f.

We first claim that there exists $v' \in C$ such that $g(v') = v_t$ and $d(v, v') = \lfloor \frac{d_r(v,v')+d_n(v,v')}{2} \rfloor$. To show this, let $v'_k \in C$ be the vertex k units above the lower binary tree such that $g(v'_k) = v_t$. (Note that k is allowed to range from 0 to $r^* + x^*$.) If we can show that $d_r(v, v'_0) \leq d_n(v, v'_0)$ and $d_n(v, v'_{r^*+x^*}) \leq d_r(v, v'_{r^*+x^*})$, then because $d_r(v, v'_k)$ is monotonically increasing unit for unit with k and $d_n(v, v'_k)$ is monotonically decreasing unit for unit with k, there must exist a value for k at which $|d_r(v, v'_k) - d_n(v, v'_k)| \leq 1$; at this value, we can let $v' = v'_k$. So we get

$$\begin{aligned} d_r(v, v'_0) &\leq s^* + x^* + N + r^* + 2\log_2 |V_1| \\ &\leq s^* + x^* + 5N + r^* \leq d_n(v, v_t) + r^* + x^* v = d_n(v, v'_0), \\ d_r(v, v'_{r^*+x^*}) &\geq s^* + x^* + N + r^* + 2 + r^* + x^* \\ &= s^* + x^* + N + r^* + 2 + (D_{G_1} + 1)(x^* + 2s^* + 2N) + x^* \\ &\geq d_n(v, v_t) = d_n(v, v'_{r^*+x^*}). \end{aligned}$$

By the above discussion and Lemma 7, we have

$$\lfloor \frac{d_r(w, w') + d_n(w, w')}{2} \rfloor = d(w, w') \geq d(v, v') = \lfloor \frac{d_r(v, v') + d_n(v, v')}{2} \rfloor.$$

Therefore, we have that

$$d_r(w, w') + d_n(w, w') \geq d_r(v, v') + d_n(v, v') - 1.$$

Note that in this case the first vertex in V_1 in a path realized by $d_n(v, v_t)$ must be in $V_1 - V$ because G_1 is bipartite. We can now lower bound the value of $d_r(w, w') + d_n(w, w')$ as follows. This leads to the following lower bound.

$$\begin{aligned} &d_r(w, w') + d_n(w, w') \\ &\geq d_r(v, v') + d_n(v, v') - 1 = d_r(v, v_t) + d_n(v, v_t) - 1 \\ &\geq s^* + N + x^* + r^* + 2 + r^* + x^* + d_n(v, v_t) - 1 \\ &> s^* + 2x^* + 2r^* + N + d_n(v, v_t) \\ &\geq s^* + 2x^* + 2r^* + N + s^* + x^* + (2 + 2s^* + x^*)(D_{G_1} - 1) + N \\ &= L + 2x^* - 2ND_{G_1} + 2D_{G_1} - 2N - 2 = L + 2D_{G_1} + 4N - 2 \\ &\geq L + 6 \end{aligned}$$

Hence, $d(w, w') = \lfloor (d_r(w, w') + d_n(w, w'))/2 \rfloor$ is at most $L/2$ for a Case 5b graph and greater than $L/2$ for a Case 5a graph.
Moreover, the value of $d^*(v_{\text{root}}, w)$ is the same value $\lceil \log_2 |V_1| \rceil + r^* + N + s^* + x^* + 2$ in either case. Hence, it is possible to differentiate between the two possible cases by running the minimal-time solution on the graph $G''(x^*, s^*, r^*)$ and noting which regime the firing time falls into. This in turn will yield the value of D_G as described above.

Time Analysis

As we have already noticed, the time to check that G is strongly connected and $D \geq 2$ is at most $O(n + E)$. Now we estimate the time for the simulation of the minimal-time solution.

For given values of x and s, the number of vertices in $G'(x, s)$ is $\tilde{O}(E(x + s))$. Note that throughout the algorithm $x \leq s$ and therefore the number of vertices in $G'(x, s)$ can be expressed as $\tilde{O}(Es)$. The number of vertices in the connectors of $G''(x, s, r)$ is $\tilde{O}((E+n)(r+x))$, and the number of vertices in the lower binary tree is $\tilde{O}(n)$. Hence, the total number of vertices in $G''(x, s, r)$ is $\tilde{O}((E + n)(r + x) + Es)$. Each vertex has degree at most 3 by construction and so the total number of edges is asymptotically this same expression $\tilde{O}((E + n)(r + x) + Es)$. Note that throughout the algorithm $s \leq r$, and because G is assumed to be connected $n \leq E + 1$. Then the total number of edges in $G''(x, s, r)$ can be expressed as $\tilde{O}(Er)$.

By Lemma 1 the minimum firing time of $G''(x, s, r)$ is $\tilde{O}(sD_G + r) = \tilde{O}(rD_G)$. Hence the time for a simulation on $G''(x, s, r)$ is $\tilde{O}(Er^2 D_G)$. Because s and r are increased geometrically, the total simulation time is asymptotically dominated by the final run on $G''(x^*, s^*, r^*)$. At that point $r^* = \tilde{O}(D_G^2)$. Thus, the asymptotic time for the total simulation is $\tilde{O}(D_G^5 E)$.

References

1. Goldstein, D., Kobayashi, K.: On the complexity of network synchronization. SIAM J. Comput. 35(3), 567–589 (2005)
2. Goldstein, D., Kobayashi, K.: On the complexity of the "most general" firing squad synchronization problem. In: Durand, B., Thomas, W. (eds.) STACS 2006. LNCS, vol. 3884, pp. 696–711. Springer, Heidelberg (2006)
3. Kobayashi, K.: On time optimal solutions of the firing squad synchronization problem for two-dimensional paths. Theoretical Computer Science 259, 129–143 (2001)
4. Mazoyer, J.: An overview of the firing synchronization problem. In: Choffrut, C. (ed.) Automata Networks. LNCS, vol. 316, pp. 12–16. Springer, Heidelberg (1988)

Capacitated Domination Problem*

Mong-Jen Kao and Chung-Shou Liao

Department of Computer Science and Information Engineering,
National Taiwan University, Taipei, Taiwan
{r95078,d93018}@csie.ntu.edu.tw

Abstract. We consider a generalization of the well-known domination problem on graphs. The (*soft*) capacitated domination problem with demand constraints is to find a dominating set D of minimum cardinality satisfying both the capacity and demand constraints. The capacity constraint specifies that each vertex has a capacity that it can use to meet the demand of dominated vertices in its closed neighborhood, and the number of copies of each vertex allowed in D is unbounded. The demand constraint specifies that the demand of each vertex in V is met by the capacities of vertices in D dominating it. In this paper, we study the capacitated domination problem on trees. We present a linear time algorithm for the unsplittable demand model, and a pseudo-polynomial time algorithm for the splittable demand model. In addition, we show that the capacitated domination problem on trees with splittable demand constraints is NP-complete (even for its integer version) and provide a $\frac{3}{2}$-approximation algorithm. We also give a primal-dual approximation algorithm for the weighted capacitated domination problem with splittable demand constraints on general graphs.

1 Introduction

The domination problem on graphs is one of the well-known combinatorial optimization problems. The domination problem can be described as follows. Let $G = (V, E)$ denote an undirected graph with vertex set V and edge set E. G is a weighted graph if each vertex $v \in V$ is associated with a weight $w(v) \in \mathbb{R}^+$. A vertex v is said to *dominate* itself and each of its neighbors. A set $D \subseteq V$ is called a *dominating set* if every vertex in V is dominated by at least one vertex in D. The goal is to find an optimal dominating set D^* of minimum weight in G. The weight of D^* is equal to the sum of the weights of all the vertices in D^*, and the minimum weight of D^* is called the *weighted domination number* of G, denoted $\gamma_w(G)$. If the weight of each vertex is unity, then $\gamma_w(G)$ would be the cardinality of the optimal dominating set D^*, and it is called the *domination number* of G, denoted $\gamma(G)$.

* Supported in part by the National Science Council under the Grants NSC95-2221-E-001-016-MY3, NSC-94-2422-H-001-0001, and NSC-95-2752-E-002-005-PAE, and by the Taiwan Information Security Center (TWISC) under the Grants NSC NSC95-2218-E-001-001, NSC95-3114-P-001-002-Y, NSC94-3114-P-001-003-Y and NSC 94-3114-P-011-001.

T. Tokuyama (Ed.): ISAAC 2007, LNCS 4835, pp. 256–267, 2007.

There has been considerable amount of research devoted to the metric case of the *capacitated facility location* problem. The capacitated facility location problem is defined as follows. Consider a set C of clients and a set F of facilities. Each client has associated with it a *demand* and each facility has a *capacity* that specifies the *maximum* service the facility can provide to its clients to meet their demands. In addition a facility also has a *setup* or *operating cost* if it is opened and *service* cost, which is based on a predefined function $\mathcal{A} : F \times C \rightarrow \mathbb{R}^+$, among pairs of facilities and clients, where $\mathbb{R}^+$ denotes the set of non-negative real numbers, and $\mathcal{A}(f, c), f \in F, c \in C$, denotes the service cost when client c is assigned to be serviced by facility f. The metric service function $\mathcal{A}$ is nonnegative and symmetric, and obeys the triangle inequality. The goal is to find a subset $K \subseteq F$ of facilities to set up and an assignment of clients to facilities, such that the demand requirements of all the clients are satisfied, facilities capacities are not violated, and the total cost, including facility setup and service cost, is minimized. The capacity of a facility in the capacitated facility location problem is said to be *hard* if the facility can be opened at most a certain *limited* number of times to serve clients' demands; it is said to be *soft* if the facility can be opened without any restrictions on the number of times it can be opened. In addition, the demand of a client is called *unsplittable* if we require that the entire demand of the client is served by a *single* facility; otherwise, we call the demand *splittable*. In general, for each client v, let the *maximum* number of distinct facilities to serve a client v be called the *demand split number* of v, denoted k_v.

The *soft* capacitated facility location problem was considered in the literature [1,13,14,22,24] and approximation algorithms based on some linear programming techniques, like LP-rounding and primal-dual algorithms were obtained. In 1997 Shmoys et al.[24] first presented a 5.69-approximation algorithm for the soft uniform capacity model with demand constraints. Using primal-dual method Jain and Vazirani[14] gave a 4-approximation algorithm for the generalized nonuniform capacity model. Arya et al.[1] improved this approximation factor to $2 + \sqrt{3}$ based on local search heuristics. Following the method of Jain and Vazirani[14], Jain et al.[13] further improved the factor to 3. The latest result, with factor 2, due to Mahdian et al.[22], achieves the integrality gap of the natural LP relaxation of this soft capacity model. The *hard* capacitated facility location problem was treated differently however. Because of the large integrality gap, linear programming techniques do not work efficiently, and local search heuristics were proposed instead. Korupolu et al.[16] first gave a $(8 + \epsilon)$-approximation algorithm for the hard uniform capacity model. Chudak and Williamson[5] improved the approximation factor to $(6 + \epsilon)$. Based on the scaling technique, Charikar and Guha[3] improved the factor to $(3 + 2\sqrt{2})$. Levi et al.[17] further improved the factor to 5. P*á*l et al.[23] presented a $(9 + \epsilon)$-approximation algorithm for the generalized hard nonuniform capacity model. This was improved by Mahdian and P*á*l.[21], and Zhang et al.[25] to yield an approximation factor $(3 + 2\sqrt{2})$.

In this paper we investigate the capacitated domination problem on graphs, which is closely related to the capacitated facility location problem, in which

the clients and facilities are vertices of the graphs, and the set of facilities we open corresponds to the dominating set, and all the distances are either 0 or ∞. The capacitated domination problem was introduced by Haynes et al.[10], who discussed a lot of variations of domination problem. In addition, the k-tuple domination problem, i.e., to find a minimum vertex subset such that every vertex in the graph is dominated by at least k vertices in this set, which was investigated by Liao and Chang[18,19] in 2002, is slightly similar to the capacitated domination problem with demand split number $k_v = k$ for each v.

The *capacitated vertex cover problem* is another famous problem related to the capacitated domination problem on graphs. Chuzhoy and Naor[4] proved weighted capacitated vertex cover problem with hard capacity and uniform demand is at least as hard as the *set cover* problem and provided a logarithmic approximation algorithm. For unweighted version with hard capacity and unsplittable nonuniform demand, they proved that this model is unapproximable unless P=NP and gave a 3-approximation algorithm for this model when each edge demand is uniform. Gandhi et al.[7] further improved the approximation ratio 3 of the latter model to 2, which is the best known ratio for the general vertex cover problem. About the soft capacity model, Gandhi et al.[8] provided an LP-rounding 2-approximation algorithm for the soft capacitated vertex cover problem with uniform demand. Guha et al.[9] presented the same result by a primal-dual method, and a 3-approximation algorithm for the generalized model with soft capacity and splittable nonuniform demand. They also gave polynomial time algorithms for some restricted cases of soft capacitated vertex cover problem on trees.

Our Contribution

To the best of our knowledge this is the first paper considering both notions of capacity and demand in domination problem on graphs, which is as hard as *set cover* problem. In particular, we study the *soft* capacitated domination problem with demand constraints on trees. We present a linear time algorithm for the *unsplittable* demand model. For the *splittable* demand model on trees, we show it is NP-complete even when the vertex capacity and demand are integers, in contrast to the linear time result of Guha et. al[9] for the same model of capacitated vertex cover on trees. Furthermore, based on our NP-completeness reduction, we develop a pseudo-polynomial time algorithm and further a combinatorial $\frac{3}{2}$-approximation algorithm for splittable demand model on trees. We also give a primal-dual $(\Delta + 1)$-approximation algorithm for weighted capacitated domination problem with splittable demand constraints on general graphs, where Δ is the maximum degree of the vertices. The approximation factor is almost equal to one of the two well-known approximation ratios (Δ[2,11] and $O(\ln n)$[6,15,20]) of general domination problem.

2 Preliminaries

Let $G = (V, E)$ be an undirected graph with vertex set V and edge set E. A vertex $w \in V$ is said to be *a neighbor of* or *adjacent to* a vertex $v \in V$ if

$(v, w) \in E$. The *neighborhood* of a vertex $v \in V$ is $N_G(v) = \{w \in V |\ (v, w) \in E\}$. The *closed neighborhood* of $v \in V$ is $N_G[v] = N_G(v) \cup \{v\}$. The closed degree of a vertex $v \in V$ is $deg[v] = |N_G[v]|$. The closed degree of a graph G is $\Delta_G^* = \max_{v \in V} deg[v]$.

In the *capacitated domination problem with demand constraints*, each vertex $v \in V$ has a positive demand $d(v)$, required of service from vertices in $N_G[v]$, and a positive capacity $c(v)$, which is the *maximum* service v can provide to vertices in $N_G[v]$. A multi-set $D \subseteq V$, with a *multiplicity function* $x : V \to \mathbb{Z}^+ \cup \{0\}$, is called a *capacitated dominating multi-set* if there exists an assignment function $f : V \times D \to \mathbb{R}^+ \cup \{0\}$ such that for every $v \in V$ and $\delta \in D \cap V$, the following constraints $(*)$ are satisfied. Note that $f(v, \delta)$ denotes the amount of demand requirement of each vertex $v \in V$ that is assigned to a vertex $\delta \in D \cap V$, and $f(v, \delta) = 0$ if $v \notin N_G[\delta]$.

1. $\sum_{\forall \delta \in N_G[v] \cap D} f(v, \delta) \geq d(v)$ (demand constraint for v)
2. $\sum_{\forall v \in N_G[\delta]} f(v, \delta) \leq x(\delta) \times c(\delta)$ (capacity constraint for δ) $(*)$

More formally, given a graph $G = (V, E)$, a capacity function $c : V \to \mathbb{R}^+ \cup \{0\}$, and a demand function $d : V \to \mathbb{R}^+ \cup \{0\}$, the *capacitated domination problem with demand constraints* is to find a *capacitated dominating multi-set* D of G and an assignment function $f : V \times D \to \mathbb{R}^+ \cup \{0\}$, which satisfies both the demand constraint and capacity constraint (specified in $(*)$) such that $\sum_{v \in V} x(v)$ is minimum, where $x(v)$, defined by a multiplicity function $x : V \to \mathbb{Z}^+ \cup \{0\}$, denotes the number of copies of $v \in V$ that belongs to D. $\sum_{v \in V} x(v)$ is the *cardinality* or the *size* of the multi-set D, denoted $|D|$.

One can consider the weighted capacitated domination problem by associating with each vertex a positive real weight $w(v)$, which denotes the cost incurred and may differ for different vertex v. Then the problem becomes that of finding a capacitated dominating multi-set such that the total weight $\sum_{v \in V} w(v) \times x(v)$ is minimum. There are also variations in the assignment of demands for each vertex and in the selection of multiple copies of vertices. We say the demand is unsplittable if we require that $f(v, \delta)$ is either $d(v)$ or 0 for each $v \in V$ and $\delta \in N_G[v] \cap D$, and splittable if no such restriction exists on f. The capacity is said to be *soft* if the number of available copies of each vertex is unbounded and *hard* otherwise.

3 Capacitated Domination on Trees

Guha, et al.[9] showed in 2002, that the *weighted* capacitated vertex cover problem on trees is NP-hard and that the unweighted case can be solved in linear time. In this section, we reveal that there exists, however, a considerably big gap between capacitated vertex cover problem and capacitated domination problem on trees for the unweighted case when the demand is splittable.

3.1 Unsplittable Demand Model

In this subsection we show that capacitated domination problem with unsplittable demand on trees can be solved in linear time. In particular, we consider a stronger version of the original problem to find an optimal capacitated dominating multi-set D with the maximum *residue capacity* at the root. The residue capacity of a vertex δ, denoted by $RC[\delta]$, is the remainder of its capacity after δ serves the demands of vertices in $N_G[\delta]$, i.e., $RC[\delta] = x(\delta) \times c(\delta) - \sum_{\forall v \in N_G[\delta]} f(v, \delta)$. The algorithm runs in a bottom-up manner. It processes one vertex at each iteration in postorder tree traversal. Since the demand is unsplittable, we will use the words *assign the demand of v to δ* or *assign v to δ* alternatively for convenience.

Given a tree $T = (V, E)$ with a postorder tree traversal, let $p(v)$ be the parent of $v \in V$, $Ch(v)$ the child set of v, and T_v the subtree rooted at v. At each iteration i, we compute the minimum cardinality of the capacitated dominating multi-set and the residue capacity of v_i for T_{v_i} when v_i assigned to itself or to one of its children in $Ch(v_i)$ or when v_i assigned to its parent $p(v_i)$. The minimum cardinality of the capacitated dominating multi-set and the residue capacity of v_i in the former case are denoted $W_{\downarrow}[v_i]$ and $RC_{\downarrow}[v_i]$ respectively and in the latter they are denoted $W_{\uparrow}[v_i]$ and $RC_{\uparrow}[v_i]$ respectively. $W_{\uparrow}[v_i]$ is assumed to be infinity if v_i is the root of T. Note that in the former case when there is more than one child of v_i providing minimum $W_{\downarrow}[v_i]$, we select the one with the maximum $RC_{\downarrow}[v_i]$, and in case of a tie, we break it arbitrarily. Due to space constraint, we shall skip the proofs. More details can be found in Kao et al.[26].

Lemma 1. *Given a postorder tree traversal $v_1, v_2, \ldots, v_n$, we can at each iteration i, $1 \leq i \leq n$, immediately determine the assignment of the demand of v_i optimally to its parent $p(v_i)$ or otherwise, except for the case when $W_{\downarrow}[v_i] = W_{\uparrow}[v_i]$ and $RC_{\downarrow}[v_i] > RC_{\uparrow}[v_i]$ (we call this case* **undetermined condition** *of v_i).*

The main idea of our algorithm, ALGORITHM MCDUT (see [26]), is described in the following. Given a tree $T = (V, E)$ with a postorder tree traversal $v_1, v_2, \ldots, v_n$, we shall process vertices one at a time in postorder. We maintain four variables, $W_{\uparrow}[v_i]$, $W_{\downarrow}[v_i]$, $RC_{\uparrow}[v_i]$, $RC_{\downarrow}[v_i]$, and one pointer from v_i to v_j, if necessary, where v_j is the child of v_i which gives the optimal $W_{\downarrow}[v_i]$, among all children of v_i. For convenience we assume that the capacity of the parent of v_i, $c(p(v_i))$ is available when we process v_i. At each iteration i, we do the following computations.

Case(1). Computation of $W_{\uparrow}[v_i]$ and $RC_{\uparrow}[v_i]$.

If v_i is to be assigned to $p(v_i)$, we compute $W_{\uparrow}[v_i]$ as (1) the number of copies of $p(v_i)$ needed, i.e., $\lceil \frac{d(v_i)}{c(p(v_i))} \rceil$, plus (2) the summation of $\min\{W_{\downarrow}[u], W_{\uparrow}[u]\}$ for every child u of v_i, and minus (3) the saving of copies of v_i , i.e., $\lfloor \frac{RC[v_i]}{c(v_i)} \rfloor$, provided by the residue capacity of v_i due to the assignments of its children, if any, where $RC[v_i] = \sum_{u \in Ch(v_i), W_{\downarrow}[u] \geq W_{\uparrow}[u]} (c(v_i) - (d(u) \bmod c(v_i)))$.

We compute $RC_{\uparrow}[v_i]$ as $RC[v_i] \bmod c(v_i)$.

Case(2). Computation of $W_{\downarrow}[v_i]$ and $RC_{\downarrow}[v_i]$.

Case(2-1). If v_i is assigned to itself, we compute $W_{\downarrow}[v_i]$ in a way similar to Case(1) with some modifications: part (1) is modified to be the number of copies of v_i, i.e., $\lceil \frac{d(v_i)}{c(v_i)} \rceil$. The computation of part (2) is the same. The computation of part (3) the savings of copies of v_i, is the same as before, i.e., $\lfloor \frac{RC[v_i]}{c(v_i)} \rfloor$, except that $RC[v_i]$ given above needs to be adjusted by adding $(c(v_i) - (d(v_i) \bmod c(v_i)))$.

Case(2-2). If v_i is assigned to one of its children, say $u^* \in Ch(v_i)$, we compute $W_{\downarrow}[v_i]$ in a way similar to Case(1) with the following modifications: part (1) is modified to be the number of copies of u^*, i.e., $\lceil \frac{d(v_i)}{c(u^*)} \rceil$. The computation of part (2) is the same as above and the computation of part (3) is modified, depending on the status of u^*.

Case(2-2-1) assignment of u^* is determined. We add the saving of copies of u^*, $\lfloor (RC[u^*] + (c(u^*) - (d(v_i) \bmod c(u^*))))/c(u^*) \rfloor$, to part (3), where $RC[u^*] = RC_{\uparrow}[u^*]$ if u^* was assigned upward by Lemma 1; and $RC[u^*] = RC_{\downarrow}[u^*]$, if u^* was assigned downward.

Case(2-2-2) assignment of u^* is undetermined. In this case, $W_{\uparrow}[u^*] = W_{\downarrow}[u^*]$ and $RC_{\uparrow}[u^*] < RC_{\downarrow}[u^*]$. We need to consider both cases in which u^* is assigned upward and downward, in order to obtain the optimal $W_{\downarrow}[v_i]$. The case when u^* is assigned upward is the same as Case(2-2-1). For the case when u^* is assigned downward, we need to modify $RC[v_i]$ given in Case (1) by subtracting from it $(c(v_i) - (d(u^*) \bmod c(v_i)))$ and then compute for part (3) $\lfloor \frac{RC[v_i]}{c(v_i)} \rfloor + \lfloor (RC_{\downarrow}[u^*] + (c(u^*) - (d(v_i) \bmod c(u^*))))/c(u^*) \rfloor$.

We compute $RC_{\downarrow}[v_i]$ as $RC[v_i] \bmod c(v_i)$ for each subcase Case(2-1) and Case(2-2).

We shall select as the optimal $W_{\downarrow}[v_i]$ the assignment for v_i that gives the minimum $W_{\downarrow}[v_i]$ with the maximum $RC_{\downarrow}[v_i]$ among all the cases. In addition, if the optimal $W_{\downarrow}[v_i]$ is obtained from the assignment of v_i to a child $\hat{u}$, we add a pointer from v_i to $\hat{u}$, and furthermore if $\hat{u}$ was undetermined, we record which assignment, upward or downward, of $\hat{u}$ in Case(2-2-2) yields the optimal $W_{\downarrow}[v_i]$.

Lemma 2. *If v_i is classified as undetermined at iteration i, then at most one child of v_i remains undetermined. Otherwise, if v_i is classified as determined at iteration i, then each child of v_i is determined as well.*

Corollary 1. (Invariant Condition) *After each iteration i, $1 \leq i \leq n$, there is at most one path consisting of all undetermined vertices in T_{v_i} starting with v_i.*

We remark that Algorithm MCDUT[26] can be modified to output the optimal capacitated dominating multi-set $(x(\delta) = \lceil \frac{\sum_{v \in N[\delta]} f(v,\delta)}{c(\delta)} \rceil, \forall \delta \in D)$ as well.

Theorem 1. Algorithm MCDUT *solves the capacitated domination problem with unsplittable demand on trees in linear time.*

3.2 Splittable Demand Model

We next consider the case when the vertex demand is splittable. We first show the problem is **NP-complete** even when the vertex capacity and demand are integers. We shall reduce the decision problem *Subset Sum*, which is a well-known NP-complete problem, to the decision version of capacitated domination problem with splittable demand on trees. The Subset Sum problem is defined as: Given a finite set $S = \{a_1, a_2, \ldots, a_n\}$, $\forall a_i \in \mathbb{Z}^+$, and $W \in \mathbb{Z}^+$, the decision Subset Sum problem is to determine if there exists a subset A of $\{1, 2, \ldots n\}$ such that $\sum_{i \in A} a_i = W$. Without loss of generality, we assume that $\sum_{i=1}^{n} a_i \geq W$.

Let $M = (\sum_{i=1}^{n} a_i) + 1$, $W' = M \cdot W$, and $a'_i = M \cdot a_i$. Given an instance of the decision Subset Sum problem, we build a capacitated domination problem instance T, where T is a tree consisting of $n+2$ vertices, where v_{n+1} is the root with capacity 1 and demand W, v_0 is the only child of v_{n+1} with capacity M and demand $\sum_{i=1}^{n} a'_i - W'$, and v_0 has as children, $v_1, v_2, \ldots, v_n$, which are leaf vertices and each leaf v_i has capacity $M + a'_i - a_i$ and demand $M - a_i$. It is not difficult to see that there exists a subset A of $\{1, 2, \ldots n\}$ such that $\sum_{i \in A} a_i = W$ if and only if there exists a feasible capacitated dominating multi-set D of size n for T.

Theorem 2. *The capacitated domination problem with splittable demand on trees is NP-complete, even when the vertex capacity and demand are integers.*

We then present a **pseudo-polynomial time** algorithm for splittable demand model. Given a tree $T = (V, E)$ with a postorder tree traversal, at each iteration i, we maintain the *residue demand* of v_i, denoted by $RD[v_i]$, and the residue capacity of v_i, denoted by $RC[v_i]$. Initially we have $RD[v_i] = d(v_i)$ and $RC[v_i] = 0$ for every $v_i \in V$. As the vertex v_i is considered, the algorithm first uses up all the available residue capacity, if any, from $Ch(v_i)$ and then from $\{v_i\}$, attempting to assign as much $RD[v_i]$ as possible so that either $RD[v_i]$ is satisfied or all the available residue capacity is exhausted. We describe **the first step**, **exhausting all the available residue capacity** from $Ch(v_i) \cup \{v_i\}$, in detail later.

After the first step, if $RD[v_i]$ is satisfied, then $RC[v_i]$ is also maximized. Otherwise, if $RD[v_i] \neq 0$, that is, the available residue capacity is used up but $RD[v_i]$ is not satisfied, we process $RD[v_i]$ in a greedy manner as follows. Find $u^* \in N[v_i]$ such that $c(u^*)$ is $\max_{u \in N[v_i]}\{c(u)\}$. Create $\lfloor RD[v_i]/c(u^*) \rfloor$ copies of $u^* \in D$ to meet the demand $RD[v_i]$ of v_i. We then have for v_i a new residue demand $RD[v_i]$ which is $RD[v_i]$ mod $c(u^*)$. The algorithm further decides whether the new $RD[v_i]$ can be determined immediately. We recall that Lemma 1 characterizes the *undetermined condition* at each iteration i in the unsplittable demand model. A similar *undetermined condition* also holds in the splittable demand model.

Lemma 3. *Given a tree $T = (V, E)$ with a postorder tree traversal $v_1, v_2, \ldots, v_n$, we can at each iteration i, $1 \leq i \leq n$, immediately determine the optimal assignment of the residue demand $RD[v_i]$ of v_i, except for the case when $RD[v_i] \leq c(p(v_i)) < c(v_i)$ (this is called the* **undetermined condition** *of v_i).*

If v_i can be determined immediately, then we have $RD[v_i] = 0$. Otherwise we have $RC[v_i] = 0$ by the first step and mark v_i as *undetermined*. The two invariant conditions hold after each iteration i, $1 \leq i \leq n$, (1) $RD[v_i] \cdot RC[v_i] = 0$; and (2) $RD[v_i] \neq 0$ only if $RD[v_i] \leq c(p(v_i)) < c(v_i)$. Now we describe **the first step**, **exhausting all the available residue capacity** from $Ch(v_i) \cup \{v_i\}$, of our algorithm in detail. When the vertex v_i is considered, the algorithm exhausts the available residue capacity from $Ch(v_i)$ first. Let $Ch_d(v_i) = \{p_1, p_2, \ldots, p_l\}$ be the set of *determined* children of v_i and $Ch_u(v_i) = \{q_1, q_2, \ldots, q_m\}$ be the set of *undetermined* children of v_i. Note that $RD[p_j] = 0$, $1 \leq j \leq l$, and $RD[q_k] \leq c(v_i) < c(q_k)$, $1 \leq k \leq m$. The first step has the following two phases.

Phase(1). *Exhaust the total residue capacity in* $Ch_d(v_i)$*, i.e.,* $\sum_{p_j \in Ch_d(v_i)} RC[p_j]$. We assign $RD[v_i]$ to $p_1, \ldots, p_l$ as far as possible. If $RD[v_i] \leq \sum_{p_j \in Ch_d(v_i)} RC[p_j]$, then $RD[v_i]$ can be satisfied. We then assign $q_1, q_2, \ldots, q_m$ upward to v_i since it provides $p(v_i)$ more residue capacity, and the first step is done. Otherwise, if $RD[v_i] > \sum_{p_j \in Ch_d(v_i)} RC[p_j]$, we update $RD[v_i] = RD[v_i] - \sum_{p_j \in Ch_d(v_i)} RC[p_j]$ and enter the next phase.

Phase(2). *Arrange the demand assignment of the vertices in* $Ch_u(v_i)$ *to satisfy* $RD[v_i]$ *while* $RC[v_i]$ *is maximized.*
If $RD[v_i] > \sum_{q_k \in Ch_u(v_i)} (c(q_k) - RD[q_k])$, then it is impossible that $RD[v_i]$ can be satisfied by any arrangement of the total available residue capacity of $Ch_u(v_i)$, since by the undetermined condition of q_k, $c(q_k) > c(v_i)$ for each $q_k \in Ch_u(v_i)$ and we assign $q_1, q_2, \ldots, q_m$ downward to themselves. Subsequently, we use up all available residue capacity in $Ch_u(v_i)$ to satisfy $RD[v_i]$, and then assign the rest of $RD[v_i]$, $RD[v_i] - \sum_{q_k \in Ch_u(v_i)} (c(q_k) - RD[q_k])$, to itself with the residue capacity of v_i, $RC[v_i]$. This completes the **first step**.

Otherwise, i.e., $RD[v_i] \leq \sum_{q_k \in Ch_u(v_i)} (c(q_k) - RD[q_k])$, we will proceed to arrange the demand assignment of the vertices in $Ch_u(v_i)$ to satisfy $RD[v_i]$ while maximizing $RC[v_i]$. This problem reduces to the following Relaxed Knapsack Problem, which is a variation of the well-known *Knapsack Problem*. Note that the first invariant condition, that is, $RD[v_i]$ is satisfied or $RC[v_i]$ is exhausted, holds in both phases.

Definition 1. (The Relaxed Knapsack Problem) *Given a set of* m *ordered pairs* (a_k, b_k) *denoting respectively the size and profit of the* k^{th} *item,* $\forall a_k, b_k \in \mathbb{Z}^+ \cup \{0\}$, $1 \leq k \leq m$, *and a nonnegative integer* W, *find a subset* $A \subseteq \{1, 2, \ldots, m\}$ *such that* $\sum_{k \in A} b_k - \max\{0, \sum_{k \in A} a_k - W\}$ *is maximized.*

We select the items and place them into a knapsack of the relaxed size W to maximize the sum of the profit $\sum_{k \in A} b_k$. In particular, the additional compensation $\sum_{k \in A} a_k - W$ is required if the total size of the selected items, $\sum_{k \in A} a_k$, exceeds W. The Relaxed Knapsack Problem can be solved in $O(m^2 M)$ pseudo-polynomial time[26], where $M = \max_{1 \leq k \leq m} \{b_k\}$.

Theorem 3. *The demand assignment of vertices in $Ch_u(v_i)$ can be optimally determined in $O(|Ch_u(v_i)|^2 c(v_i))$ time such that $RD[v_i]$ is satisfied while maximizing $RC[v_i]$ for Phase(2) of the first step at each iteration i, $1 \leq i \leq n$.*

Corollary 2. *There exists at most one undetermined vertex, namely v_i, in T_{v_i}, after each iteration i, $1 \leq i \leq n$.*

Theorem 4. *Given a tree $T = (V, E)$ with a postorder tree traversal, the capacitated domination problem with splittable demand on T can be solved in $O(C|V|^2)$ time, where $C = \max_{v_i \in V} c(v_i)$.*

4 Approximation Algorithms

4.1 $\frac{3}{2}$-approximation on Trees

In this subsection, we first give a fully polynomial time approximation scheme (FPTAS) for the Relaxed Knapsack Problem. The idea is similar to the well-known FPTAS result of Ibarra and Kim[12] for the *Knapsack Problem*. Based on our FPTAS result, we then present a simple $\frac{3}{2}$-approximation algorithm for the capacitated domination problem with splittable demand on trees. We remark that our $\frac{3}{2}$-factor approximation for trees can be improved to a polynomial time approximation scheme in a similar manner.

Algorithm FPTAS_RKP (FPTAS for Relaxed Knapsack Problem)

1. Given $\epsilon > 0$, let $k = \frac{\epsilon M}{2n+1}$, $a'_i = \lceil \frac{a_i}{k} \rceil$, $b'_i = \lfloor \frac{b_i}{k} \rfloor$, and $W' = \lfloor \frac{W}{k} \rfloor$, where $M = \max_{1 \leq i \leq n} \{b_i\}$.
2. Use (a'_i, b'_i), $1 \leq i \leq n$, and W' as input parameters of the new Relaxed Knapsack Problem instance. Apply the dynamic programming method given in [26] to obtain a max profit B^*.
3. Output the solution A^* by the backtracking technique from B^*.

Lemma 4. *Let A^* be the solution obtained by Algorithm* FPTAS_RKP. *We have profit$(A^*) \geq (1 - \epsilon) \cdot$ profit(O), where O is the original optimal solution.*

Theorem 5. *Algorithm* FPTAS_RKP *is an FPTAS for* Relaxed Knapsack Problem *which gives a $(1-\epsilon)$-approximation in $O(n^2 \lfloor \frac{M}{k} \rfloor)$ or $O(n^2 \lfloor \frac{n}{\epsilon} \rfloor)$ time.*

Based on our FPTAS result of the Relaxed Knapsack Problem, we provide a simple $\frac{3}{2}$-approximation algorithm for the capacitated domination problem with splittable demand on a tree $T = (V, E)$ in $O(|V|^3 \Delta)$ time, where Δ is the maximum degree, as follows. Given a capacitated domination problem instance, let $\epsilon = \frac{1}{\Delta}$. Consider Phase(2) of our first step for every $v_i \in V$ in subsection 3.2. We first verify whether one copy of v_i, i.e., the capacity $c(v_i)$ is sufficient for the residue demand of v_i and the total residue demand in $Ch_u(v_i)$. That is, we place all the items into the knapsack, i.e., $A^* = Ch_u(v_i)$ in the Relaxed Knapsack Problem, and $c(v_i) \geq RD[v_i] + \sum_{q_k \in Ch_u(v_i)} RD[q_k]$. If it is true, we just have

a copy of v_i. If not, we have a copy of v_j when $Ch(v_i) = Ch_u(v_i) = \{v_j\}$. Otherwise, we use Algorithm FPTAS_RKP as above for Phase(2) of our first step to arrange the demand assignment of vertices in $Ch_u(v_i)$, and have an additional copy of v_i after the arrangement.

Theorem 6. *The approximation algorithm based on* FPTAS_RKP *yields a* $\frac{3}{2}$*-approximation factor for the capacitated domination problem with splittable demand on a tree* $T = (V, E)$ *in* $O(|V|^3\Delta)$ *time, where* Δ *is the maximum degree.*

4.2 Δ^*-approximation on General Graphs

In this subsection, we present a primal-dual algorithm[4,9,14] that gives a Δ^*-approximation for weighted capacitated domination with splittable demand on general graphs, where Δ^* is the closed degree of the input graph. The algorithm is based on the dual fitting technique.

The primal integer linear program (ILP) and its dual are given in Eq. (1) and Eq. (2) respectively below. A feasible primal solution corresponds to a feasible capacitated dominating multi-set on a given weighted graph. The additional constraint $d(v_j)x(v_i) - f(v_j, v_i) \geq 0$, which is unnecessary in the ILP formulation, is required to bound the integrality gap between the fractional optimum and the integral optimum in the relaxation. The objective value of a dual feasible solution to Eq. (2) is a lower bound of any integral optimum of the primal program.

$$
\begin{aligned}
&\text{Minimize} \quad \textstyle\sum_{i=1}^{n} w(v_i)x(v_i) \\
&c(v_i)x(v_i) - \textstyle\sum_{v_j \in N[v_i]} f(v_j, v_i) \geq 0, && \forall i. \\
&\textstyle\sum_{v_j \in N[v_i]} f(v_i, v_j) \geq d(v_i), && \forall i. \\
&d(v_j)x(v_i) - f(v_j, v_i) \geq 0, && v_j \in N[v_i]. \\
&x(v_i) \in \mathbb{Z}^+ \cup \{0\}, && \forall i.
\end{aligned}
$$

Eq. (1)

$$
\begin{aligned}
&\text{Maximize} \quad \textstyle\sum_{i=1}^{n} d(v_i)y_i \\
&c(v_i)z_i + \textstyle\sum_{v_j \in N[v_i]} d(v_j)g_{i,j} \leq w(v_i), && \forall i. \\
&y_i \leq z_j + g_{j,i}, && v_j \in N[v_i], \forall i. \\
&y_i \geq 0, \quad g_{j,i} \geq 0, && v_j \in N[v_i], \forall i.
\end{aligned}
$$

Eq. (2)

Given a weighted graph $G = (V, E)$, let $V^\phi \subseteq V$ denote the set of vertices whose demand has not been assigned yet. For every vertex v, we use $d_N^\phi(v)$ to denote the total amount of unassigned demand of all the closed neighbors of v. In addition, we define the inequality $d_N^\phi(v) \leq c(v)$ to be the *critical condition* of v and $D_{u\bar{v}}^c$ to be the amount of demand of a closed neighbor u of v, $d(u)$, which is to be assigned to vertices other than v after the critical condition of v is met. Our Algorithm MCDAWG (see [26]) works as follows. In the dual program, all the dual variables y_i are zero initially. This is a dual feasible solution with all $z_j = 0$ and $g_{j,i} = 0$. We increase all the dual variables y_i simultaneously, for each vertex $v_i \in V^\phi$ whose demand is unassigned, i.e., increase z_j or $g_{j,i}$, for every $v_j \in N[v_i]$. To be more precise, if the closed neighbors of v_j have a large amount of unassigned demands, specifically, $d_N^\phi(v_j) > c(v_j)$, then we increase z_j. Otherwise, we increase $g_{j,i}$. When we increase the dual variables simultaneously

for each vertex in V^ϕ, we stop increasing v_k as soon as the weight constraint of v_k, $c(v_k)z_k + \sum_{v_\ell \in N[v_k]} d(v_\ell)g_{k,\ell} \leq w(v_k)$, is met with equality. We then mark vertex v_k *open* and assign to it $d_N^\phi(v_k)$ unassigned demand from the closed neighbors of v_k. In addition, if the critical condition of v_k, i.e., $d_N^\phi(v_k) \leq c(v_k)$, holds, we also reassign to v_k $D^c_{u\bar{v}_k}$ demand from every closed neighbor u of v_k. Note that for every closed neighbor u of v_k, we have to update $D^c_{u\bar{w}}, \forall w \in N[u]$. All the demand assignments can be arranged accordingly, and finally the capacitated dominating multi-set is $D = \{v;\ v \text{ is marked open.}\}$ and $x(v) = \lceil (\sum_{u \in N[v]} f(u,v))/c(v) \rceil$, $\forall v \in D$.

Lemma 5. *The total weight of each open vertex, $\sum_{v \in D} w(v) \cdot x(v)$, can be distributed so that each unit of the demand (say from v_j) costs at most $deg[v_j] \cdot y_j$ weight.*

Theorem 7. ALGORITHM MCDAWG *obtains a Δ^*-approximation solution for weighted capacitated domination problem with splittable demand on a general graph $G = (V, E)$, where Δ^* is the closed degree of G.*

5 Conclusion

In this paper we have presented a linear time algorithm for capacitated domination problem with unsplittable demand on trees. As for the splittable demand model on trees, we have shown that it is NP-complete, and solvable in pseudo-polynomial time. We have also provided two approximation algorithms. One is $\frac{3}{2}$-factor for trees and the other is Δ^*-factor by using primal-dual method for general weighted graphs, where Δ^* is the closed degree of the input graph. We remark that our $\frac{3}{2}$-factor approximation for trees can be improved to a polynomial time approximation scheme in a similar manner.

References

1. Arya, V., Garg, N., Khandekar, R., Meyerson, A., Munagala, K., Pandit, V.: Local search heuristics for k-median and facility location problems. SIAM Journal on Computing 33(3), 544–562 (2004)
2. Bar-Yehuda, R., Even, S.: A linear-time approximation algorithm for the weighted vertex cover problem. Journal of Algorithms 2, 198–203 (1981)
3. Charikar, M., Guha, S.: Improved combinatorial algorithms for facility location and k-median problems. In: Proceedings of 40th IEEE Symposium of Foundations of Computer Science, pp. 378–388 (1999)
4. Chuzhoy, J., Naor, J.S.: Covering problems with hard capacities. SIAM Journal on Computing 36(2), 498–515 (2006)
5. Chudak, F.A., Williamson, D.P.: Improved approximation algorithms for capacitated facility location problems. Mathematical Programming 102(2), 207–222 (2005)
6. Chvátal, V.: A greedy heuristic for the set-covering problem. Mathematics of Operations Research 4(3), 233–235 (1979)

7. Gandhi, R., Halperin, E., Khuller, S., Kortsarz, G., Srinivasan, A.: An improved approximation algorithm for vertex cover with hard capacities. Journal of Computer and System Sciences 72(1), 16–33 (2006)
8. Gandhi, R., Khuller, S., Parthasarathy, S., Srinivasan, A.: Dependent rounding and its applications to approximation algorithms. Journal of the ACM 53(3), 324–360 (2006)
9. Guha, S., Hassin, R., Khuller, S., Or, E.: Capacitated vertex covering. Journal of Algorithms 48(1), 257–270 (2003)
10. Haynes, T.W., Hedetniemi, S.T., Slater, P.J.: Domination in Graphs: The Theory. Marcel Dekker, Inc., New York (1998)
11. Hochbaum, D.S.: Approximation algorithms for the set covering and vertex cover problems. SIAM Journal on Computing 11(3), 555–556 (1982)
12. Ibarra, O.H., Kim, C.E.: Fast approximation algorithms for the knapsack and sum of subset problems. Journal of the ACM 22(4), 463–468 (1975)
13. Jain, K., Mahdian, M., Saberi, A.: A new greedy approach for facility location problems. In: Proceedings of 34th ACM Symposium on Theory of Computing, pp. 731–740 (2002)
14. Jain, K., Vazirani, V.V.: Approximation algorithms for metric facility location and k-median problems using the primal-dual schema and Lagrangian relaxation. Journal of the ACM 48(2), 274–296 (2001)
15. Johnson, D.S.: Approximation algorithms for combinatorial problems. Journal of Computer and System Sciences 9(3), 256–278 (1974)
16. Korupolu, M., Plaxton, C., Rajaraman, R.: Analysis of a local search heuristic for facility location problems. Journal of Algorithms 37(1), 146–188 (2000)
17. Levi, R., Shmoys, D.B., Swamy, C.: LP-based approximation algorithms for capacitated facility location. In: Proceedings of 10th Conference on Integer Programming and Combinatorial Optimization, pp. 206–218 (2004)
18. Liao, C.S., Chang, G.J.: Algorithmic aspect of k-tuple domination in graphs. Taiwanese J. Math. 6, 415–420 (2002)
19. Liao, C.S., Chang, G.J.: k-tuple domination in graphs. Inform. Process. Letter. 87(1), 45–50 (2003)
20. Lovász, L.: On the ratio of optimal and fractional covers. Discrete Math. 13, 383–390 (1975)
21. Mahdian, M., Pál, M.: Universal facility location. In: Proceedings of 11th European Symposium on Algorithms, pp. 409–421 (2003)
22. Mahdian, M., Ye, Y., Zhang, J.: Approximation algorithms for metric facility location problems. SIAM Journal on Computing 36(2), 411–432 (2006)
23. Pál, M., Tardos, É., Wexler, T.: Facility location with nonuniform hard capacities. In: Proceedings of 42th Symposium of Foundations of Computer Science, pp. 329–338 (2001)
24. Shmoys, D.B., Tardos, É., Aardal, K.: Approximation algorithms for facility location problems. In: Proceedings of 29th ACM Symposium on Theory of Computing, pp. 265–274 (1997)
25. Zhang, J., Chen, B., Ye, Y.: A multi-exchange local search algorithm for the capacitated facility location problem. Mathematics of Operations Research 30(2), 389–403 (2005)
26. Kao, M.-J., Liao, C.S., Lee, D.T.: Capacitated domination problem, manuscript (September 2007), `http://www.iis.sinica.edu.tw/~shou794/research/CDS_200709.pdf`

The Complexity of Finding Subgraphs Whose Matching Number Equals the Vertex Cover Number

Sounaka Mishra[1], Venkatesh Raman[2], Saket Saurabh[2], Somnath Sikdar[2], and C. R. Subramanian[2]

[1] Indian Institute of Technology,
Department of Mathematics, Chennai 600 036, India
sounak@iitm.ac.in
[2] The Institute of Mathematical Sciences,
Chennai 600 113, India
{vraman,saket,somnath,crs}@imsc.res.in

Abstract. The class of graphs where the size of a minimum vertex cover equals that of a maximum matching is known as König-Egerváry graphs. These graphs have been studied extensively from a graph theoretic point of view. In this paper, we introduce and study the algorithmic complexity of finding maximum König-Egerváry subgraphs of a given graph. More specifically, we look at the problem of finding a minimum number of vertices or edges to delete to make the resulting graph König-Egerváry. We show that both these versions are NP-complete and study their complexity from the points of view of approximation and parameterized complexity. En route, we point out an interesting connection between the vertex deletion version and the Above Guarantee Vertex Cover problem where one is interested in the parameterized complexity of the Vertex Cover problem when parameterized by the 'additional number of vertices' needed beyond the matching size. This connection is of independent interest and could be useful in establishing the parameterized complexity of Above Guarantee Vertex Cover problem.

1 Introduction

One of the celebrated min-max results of graph theory is the König-Egerváry theorem which states that for bipartite graphs the size of a minimum vertex cover equals that of a maximum matching. The class of graphs for which equality holds includes bipartite graphs as a proper subclass and is known as König-Egerváry graphs. König-Egerváry graphs have been studied extensively from a graph theoretic point of view. A good characterization of König-Egerváry graphs was found independently by Deming [4] and Sterboul [16]. Recently in [7], Korach, Nguyen and Peis have presented an excluded subgraph characterization of König-Egerváry graphs. In this paper, we define various problems related to finding König-Egerváry subgraphs and study their algorithmic complexity from the points of view of parameterized complexity and approximation algorithms. More precisely the problems which we study in this paper are:

T. Tokuyama (Ed.): ISAAC 2007, LNCS 4835, pp. 268–279, 2007.

- König Vertex (Edge) Deletion Set (KVDS (KEDS)): Given a graph $G = (V, E)$ and a positive integer k, does there exist $V' \subseteq V$ ($E' \subseteq E$) such that $|V'|$ ($|E'|$) $\leq k$ and $G[V - V']$ ($G' = (V, E - E')$) is a König-Egerváry graph?
- Maximum Vertex (Edge) Induced König Subgraph (MVIKS (MEIKS)): Given a graph $G = (V, E)$ and a positive integer k, does there exist $V' \subseteq V$ ($E' \subseteq E$) such that $|V'|$ ($|E'|$) $\geq k$ and $G[V']$ ($G' = (V, E')$) is a König-Egerváry graph?

The KVDS and MVIKS problems (and similarly, KEDS and MEIKS) are equivalent from the point of view of NP-completeness but differ in their approximability and parameterized complexity.

In order to explain one motivation for studying the König-Egerváry subgraph problem, we need to digress and discuss about parameterized complexity. In the framework of parameterized complexity, one deals with decision problems whose inputs consist of a pair (x, k), where k is called the parameter; the goal is to decide whether (x, k) is a yes-instance or not in time $O(f(k) \cdot |x|^{O(1)})$, where f is a function of k alone. Decision problems that admit such algorithms are called *fixed-parameter tractable* (FPT). Over the last decade or so a number of *NP*-hard problems have been shown to be fixed-parameter tractable. One of the most famous fixed-parameter tractable problems is Vertex Cover. An input to this problem is a graph $G = (V, E)$ and a positive integer k and the goal is to decide whether there exists a set of at most k vertices which covers all edges. Over the years a lot of work has been done to devise better FPT algorithms for this problem; the current best algorithm runs in time $O(1.2738^k + kn)$, where $n = |V|$ [3].

Since the size of a maximum matching is a lower bound for vertex cover, a natural generalization of the Vertex Cover problem is the following: Above Guarantee Vertex Cover (g-VC): Let $G = (V, E)$ be a graph with maximum matching size $\mu(G)$ and k a positive integer. The goal is to decide whether G admits a vertex cover of size at most $\mu(G) + k$, where k is the parameter. To the best of our knowledge, this problem was first introduced in [15]. This problem (and in general, such above guarantee problems, see [11]) seems difficult. In this paper, we show that this problem is fixed-parameter equivalent to KVDS. As a corollary, we obtain an $O(n^k)$ algorithm for g-VC. To the best of our knowledge, even this was not known before.

Our second reason for studying König-Egerváry subgraph problems is that the versions of König-Egerváry subgraph problems when the resulting graph we look for is bipartite (i.e. replace König-Egerváry in the above problem definitions by *bipartite*) are well studied in the area of approximation algorithms and parameterized complexity [9,13,14]. König-Egerváry subgraph problems are natural generalizations of bipartite subgraph problems but have not been studied algorithmically. We believe that this can trigger explorations of other questions in König-Egerváry graphs. For the rest of the paper, we use König as an abbreviation of König-Egerváry.

The remaining paper is organized as follows. In Section 2, we state some results and notation that we make use of in the rest of the paper. In Section 3, we consider vertex versions and in Section 4, the edge versions of the König Subgraph problem. We study both versions from the points of view of approximation and parameterized complexity. We conclude in Section 5 with a list of open problems.

2 Preliminaries

We will make use of the following well-known results in the rest of the paper.

Lemma 1. [7] *A graph $G = (V, E)$ is König if and only if there exists a bipartition of $V = V_1 \uplus V_2$ such that V_2 is independent and there exists a matching that saturates V_1 and crosses the cut (V_1, V_2).*

Lemma 2. [4] *Given a graph G on n vertices and m edges and a maximum matching of G, one can test whether G is König in time $O(n + m)$. If G is indeed König then one can find a minimum vertex cover of G in this time.*

Since a maximum matching can be obtained in time $O(m\sqrt{n})$ [17], we have

Lemma 3. *Let G be a graph on n vertices and m edges. One can check whether G is König and if König find a minimum vertex cover of G in time $O(m\sqrt{n})$.*

We use $\mu(G)$ and $\beta(G)$ to denote, respectively, the size of a maximum matching and minimum vertex cover of G. When the graph being referred to is clear from the context, we simply use μ and β. We sometimes use $\tau(G)$ to denote the difference $\beta(G) - \mu(G)$. We use $\kappa(G)$ to denote the size of the smallest König vertex deletion set of G. Unless otherwise stated, we will use n and m to denote, respectively, the number of vertices and the number of edges of a graph. All graphs in this paper are simple and undirected.

We now briefly introduce the necessary concepts concerning parameterized complexity. A parameterized problem is a subset of $\Sigma^* \times \mathbb{N}$, where Σ is a finite alphabet and $\mathbb{N}$ is the set of natural numbers. An instance of a parameterized problem is therefore a pair (I, k), where k is the parameter. In the framework of parameterized complexity, the running time of an algorithm is viewed as a function of two quantities: the size of the problem instance *and* the parameter. A parameterized problem is said to be *fixed parameter tractable* (fpt) if there exists an algorithm for the problem with time complexity $O(f(k) \cdot |I|^{O(1)})$, where f is a function only depending on k. The class FPT consists of all fixed parameter tractable problems.

A parameterized problem π_1 is *fixed-parameter reducible* to a parameterized problem π_2 if there exist functions $f, g : \mathbb{N} \to \mathbb{N}$, $\Phi : \Sigma^* \times \mathbb{N} \to \Sigma^*$ and a polynomial $p(\cdot)$ such that for any instance (I, k) of π_1, $(\Phi(I, k), g(k))$ is an instance of π_2 computable in time $f(k) \cdot p(|I|)$ and $(I, k) \in \pi_1$ if and only if $(\Phi(I, k), g(k)) \in \pi_2$. Two parameterized problems are *fixed-parameter equivalent* if they are fixed-parameter reducible to each other. The basic complexity class for fixed-parameter intractability is $W[1]$ as there is strong evidence that $W[1]$-hard problems are not fixed-parameter tractable. To show that a problem is $W[1]$-hard, one needs to exhibit a fixed-parameter reduction from a known $W[1]$-hard problem to the problem at hand. For more on parameterized complexity see [12].

3 Vertex Versions of the König Subgraph Problems

The problems we consider in this section are KVDS and MVIKS. Since König graphs are a "generalization" of bipartite graphs and since the Vertex Bipartization problem

is now known to be FPT [13], the parameterized complexity of the KVDS problem is very interesting.

In the next subsection, we relate the KVDS problem with another important open problem in the area of parameterized complexity, the g-VC. This also proves the NP-completeness of the KVDS problem.

3.1 König Vertex Deletion Set and Above Guarantee Vertex Cover

We begin with a result which states essentially that for the g-VC problem we may, without loss of generality, assume that the input graph has a perfect matching.

Let $G = (V, E)$ be an undirected graph and let M be a maximum matching of G. Construct $G' = (V', E')$ as follows. Let I be the independent set with respect to the matching M, that is, $I = V - V[M]$. Define $V' = V \cup \{u' : u \in I\}$ and $E' = E \cup \{\{u', v\} : \{u, v\} \in E\} \cup \{\{u, u'\} : u \in I\}$. Then $M' = M \cup \{\{u, u'\} : u \in I\}$ is a perfect matching for G'.

Theorem 1. *Let G be a graph without a perfect matching and let G' be the graph obtained by the above construction. G has a vertex cover of size $\mu(G) + k$ if and only if G' has a vertex cover of size $\mu(G') + k$.*

Proof. Let M denote a maximum matching of G, I denote the set $V(G) \setminus V[M]$ and I' denote the new set of vertices that are added in constructing G'. Clearly, $\mu(G') = \mu(G) + |I|$.

($\Rightarrow$) Let C be a vertex cover of G of size $\mu(G) + k$. Define $C' = C \cup I'$. It is easy to see that C' covers all the edges of G'. Also, $|C'| = \mu(G) + k + |I'| = \mu(G') + k$.

($\Leftarrow$) Let C' be a vertex cover of G' of size $\mu(G') + k$. Define M' to be the set of edges of the form $\{\{u, u'\} : u \in I$ and $u' \in I'\}$ such that both endpoints are in C'. One can show that $C = (C' \cap V[M]) \cup \{u \in I : \{u, u'\} \in M'\}$ is a vertex cover of G of size $\mu(G) + k$. ■

The next theorem relates the vertex cover of a graph with the König vertex deletion set.

Theorem 2. *Let G be an n-vertex graph with a perfect matching. G has a vertex cover of size at most $\frac{n}{2} + k$ if and only if G has a König vertex deletion set of size at most $2k$.*

Proof. ($\Rightarrow$) Let P be a perfect matching of G and C a vertex cover of G of size at most $\frac{n}{2} + k$. Consider the subset $M \subseteq P$ of matching edges both of whose endpoints are in C. Clearly $V[M]$ is a König vertex deletion set of G of size at most $2k$.

($\Leftarrow$) Conversely let K be a König vertex deletion set of G of size $r \leq 2k$. Then $G - K$ is a König graph on $n - r$ vertices and hence has a vertex cover C' of size at most $\frac{n-r}{2}$. Clearly $C = C' \cup K$ is a vertex cover of G of size $|C'| + |K| \leq \frac{n-r}{2} + r = \frac{n+r}{2} \leq \frac{n}{2} + k$. ■

The following corollary follows from Theorems 1 and 2 and the fact that Vertex Cover is NP-complete.

Corollary 1. *The* König Vertex Deletion Set *problem is NP-complete.*

Corollary 2. *Let G be an n-vertex graph with a perfect matching P. A minimum vertex cover of G is of size $\frac{n}{2} + k$ if and only if a minimum König vertex deletion set of G is of size $2k$. Moreover there exists an edge subset $M \subseteq P$ of size k such that $V[M]$ is a minimum König vertex deletion set of G.*

If we let $\tau(G) = \beta(G) - \mu(G)$, then the above corollary states: $\kappa(G) = 2\tau(G)$.

We have shown that for graphs with a perfect matching, the KVDS problem is fixed-parameter equivalent to the g-VC problem. It is not obvious how to check whether a graph G has a vertex cover of size $\mu(G) + k$ in time $O^*(n^k)$[1]. The following theorem shows how this may be done.

Theorem 3. *Let G be a graph on n vertices and m edges. One can determine whether G has a vertex cover of size at most $\mu(G) + k$ in time $O(m\sqrt{n} + \binom{n}{k}(m + n))$.*

Proof. If G does not have a perfect matching, construct G' as in Theorem 1 in linear time. G' has a perfect matching and at most $2n$ vertices and $2m$ edges. By Theorems 1 and 2, G has a vertex cover of size at most $\mu(G) + k$ if and only if G' has a König vertex deletion set of size at most $2k$. By Corollary 2, to check whether $\kappa(G') \leq 2k$, all we need to do is select subsets of k edges of a perfect matching of G' and for each edge set, delete the endpoints of the edges and check whether the remaining graph is König. Obtaining a perfect matching takes time $O(m\sqrt{n})$. Cycling through all possible edge sets and testing whether the remaining graph is König takes time $O(\binom{n}{k}(m + n))$ by Lemma 2. Hence the claim. ∎

For a graph G with a perfect matching, Theorem 2 relates the size of a vertex cover of G with that of a König vertex deletion set of G. For graphs without a perfect matching, we have the following result.

Theorem 4. *Let G be a graph without a perfect matching. If G has a vertex cover of size $\mu(G) + k$ then G has a König vertex deletion set of size at most $2k$. Moreover, $\tau(G) \leq \kappa(G) \leq 2\tau(G)$, where $\tau(G) = \beta(G) - \mu(G)$.*

Proof. Let M be a maximum matching of G and let C be a vertex cover of G of size $\mu(G) + k$. Define $I = V \setminus V[M]$, $C_I = C \cap I$ and M' to be the subset of M both of whose endpoints are in C. One can then verify that $V[M'] \cup C_I$ is a König vertex deletion set of G of size at most $2k$.

This shows that $\kappa(G) \leq 2\tau(G)$. To prove that $\tau(G) \leq \kappa(G)$, suppose that there exists $S \subseteq V$, $|S| < \tau(G)$, such that $G \setminus S$ is König. Then the following easily verifiable inequalities: (1) $\mu(G \setminus S) \leq \mu(G)$ and (2) $\beta(G \setminus S) \geq \beta(G) - |S| = \mu(G) + \tau(G) - |S|$ imply that $\beta(G \setminus S) > \mu(G \setminus S)$, a contradiction. ∎

3.2 Approximability Results

Chen and Kanj [2] obtain an approximation for the Vertex Cover problem on graphs with a perfect matching by reducing Vertex Cover on graphs with a perfect matching to 2-Sat and then using an approximation algorithm for Max 2-Sat. Observe that by Theorem 1, we may assume without loss of generality that the input graph has a perfect matching. We observe that using the approximation algorithm for Min 2-Sat Del gives a good approximation for $\tau(G)$ and hence for $\kappa(G)$ and thus for graphs whose vertex cover size differs by a small amount from the maximum matching size.

[1] The O^* notation suppresses polynomial terms.

We now describe the reduction to Min Weight 2-Sat Del. Recall that an instance of Min Weight 2-Sat Del is a 2-CNF formula whose clauses have weights associated with them and the question is to delete clauses of minimum total weight so that the resulting formula is satisfiable. This problem is NP-complete. Let $G = (V, E)$ be a graph with a perfect matching P. For every vertex $u \in V$, define x_u to be a Boolean variable. Let $\mathcal{F}(G, P)$ denote the Boolean formula

$$\mathcal{F}(G,P) = \bigwedge_{(u,v)\in P} (\bar{x}_u \vee \bar{x}_v) \bigwedge_{(u,v)\in E} (x_u \vee x_v).$$

The proof of the next lemma follows from Theorem 2 and the proof of Theorem 5.1 in [2].

Lemma 4. *Let $G = (V, E)$ be an n-vertex graph with a perfect matching P. Then the following three statements are equivalent:*

1. *G has a vertex cover of size $\frac{n}{2} + k$.*
2. *G has a König vertex deletion set of size $2k$.*
3. *There exists an assignment that satisfies all but at most k clauses of $\mathcal{F}(G, P)$.*

From the proof of Theorem 5.1 in [2], it follows that one can find (in polynomial time) an assignment that satisfies all but at most k clauses of the form $(\bar{x}_u \vee \bar{x}_v)$, where $(u, v) \in P$, that is, clauses that correspond to the perfect matching. This is important to our approximation algorithm that we are about to present.

We need the following result on the Min Weight 2-Sat Del problem for our approximation algorithm.

Lemma 5. [1,6] *Let Φ be an instance of* Min Weight 2-Sat Del *with n variables. One can in polynomial time obtain a solution that has weight $O(\log n \log\log n)$ times the optimal. If we are willing to allow randomness, we can obtain a solution that has weight $O(\sqrt{\log n})$ times the optimal.*

Our approximation algorithm for the g-VC and KVDS problems is presented in Figure 1. Given an n-vertex graph G with a maximum matching of size μ and a minimum vertex cover of size β, this algorithm outputs a vertex cover of G of size at most $\mu + O(\log n \log\log n)(\beta - \mu)$. From Lemmas 4 and 5, we get

Theorem 5. *Let G be a graph on n vertices with a maximum matching of size μ and a minimum vertex cover of size β. Then* Algo-Above-Guar-Vertex-Cover *finds a vertex cover of G of size $\mu + O(\log n \log\log n)(\beta - \mu)$.*

Note that $V(\mathcal{S})$ in the algorithm is actually a König vertex deletion set of G. Since $|V(\mathcal{S})| \leq O(\log n \log\log n)\ (\beta - \mu)$, we have

Theorem 6. *Given a graph G on n vertices, there exists an algorithm that approximates the König vertex deletion set of G to within a factor of $O(\log n \log\log n)$.*

Thus this algorithm approximates the deficit between the sizes of a minimum vertex cover and a maximum matching. There exists a 2-approximation algorithm for the Vertex Cover problem which simply includes all vertices of a maximum matching. It is

ALGO-ABOVE-GUAR-VERTEX-COVER

Step 1. If G does not have a perfect matching, construct G' as in Theorem 1 and set $H \leftarrow G'$; else set $H \leftarrow G$.
Step 2. Find a maximum matching M of H and construct $\mathcal{F}(H, M)$.
Step 3. Use the approximation algorithm for MIN WEIGHT 2-SAT DEL to obtain an $O(\log n \log\log n)$-approximate solution $\mathcal{S}$ for $\mathcal{F}(H, M)$, where $n = |V(H)|$.
Step 4. Obtain a minimum vertex cover C of the König graph $H - V(\mathcal{S})$, where $V(\mathcal{S})$ is the set of vertices corresponding to $\mathcal{S}$.
Step 5. If $H = G$ then return $C \cup V(\mathcal{S})$; else return $(C \cup V(\mathcal{S})) - (V(G') - V(G))$.

Fig. 1. Approximation Algorithm for ABOVE GUARANTEE VERTEX COVER

a long standing open problem to devise a polynomial time algorithm which has an approximation factor less than 2.

Our algorithm is better than any *constant* factor approximation algorithm for VERTEX COVER whenever $\beta - \mu = o(\frac{n}{\log n \log\log n})$ and $\mu = \Omega(n)$. To see this, note that a c-approximate algorithm, $c > 1$, outputs a solution of size $\mu c + (\beta - \mu)c$ whereas our algorithm outputs a solution of size $\mu + O(\alpha(\beta - \mu))$, where $\alpha = \log n \log\log n$. Now if $\beta - \mu = o(\frac{n}{\alpha})$ and if $\mu = \Omega(n)$, then our algorithm outputs a solution of size $\mu + o(\mu)$, which is better than $\beta c = \mu + \mu(c-1) + (\beta - \mu)c \geq \mu + \Omega(\mu)$.

One can obtain a randomized algorithm for the g-VC and KVDS problems using the $O(\sqrt{\log n})$-randomized approximation algorithm for MIN WEIGHT 2-SAT DEL, mentioned in Lemma 5, in Step 3 of the algorithm. In the next subsection we show that the KVDS problem cannot be approximated to within 1.7212 unless P = NP.

3.3 Hardness Results

In this subsection we show the hardness of approximating the MVIKS problem. We show this by a reduction from the INDEPENDENT SET problem to the MVIKS.

Theorem 7. *There is no approximation algorithm for MVIKS with factor $O(n^{1-\epsilon})$, for any $\epsilon > 0$, unless P = NP.*

Proof. We give a reduction from INDEPENDENT SET to the MVIKS problem. Given an instance (G, k) of INDEPENDENT SET, construct a graph H as follows. The vertex set of H consists of two copies of $V(G)$ namely, $V_1 = \{u_1 : u \in V(G)\}$ and $V_2 = \{u_2 : u \in V(G)\}$. For all $u \in V(G)$, $(u_1, u_2) \in E(H)$. If $(u, v) \in E(G)$, add the edges (u_1, v_1), (u_2, v_2), (u_1, v_2) and (v_1, u_2) in $E(H)$. H has no more edges.

We claim that G has an independent set of size k if and only if H has a König subgraph of size $2k$. Let I be an independent set of size k in G. Let $K = \{u_1, u_2 \in V(H) : u \in I\}$. Clearly $G[K]$ is an induced matching on $2k$ vertices and hence König. Conversely, let K be a König subgraph of H on $2k$ vertices. By Lemma 1, every König graph on n vertices has an independent set of size at least $n/2$. Therefore let I' be an independent set of K of size at least k. Define $I = \{u \in V(G) : \text{either } u_1 \text{ or } u_2 \in I'\}$. It is clear that the vertices of I' correspond to distinct vertices of G and hence $|I| \geq k$.

It is also easy to see that the vertices in I actually form an independent set. Since the Independent Set problem can have no approximation algorithms with factor $O(n^{1-\epsilon})$, for any $\epsilon > 0$, unless P = NP [10,18], this completes the proof. ■

Note that the reduction in Theorem 7 is a fixed parameter reduction and since Independent Set is $W[1]$-complete, we have

Corollary 3. *The parameterized version of the MVIKS problem is $W[1]$-hard.*

In the reduction above, $|V(H)| = 2|V(G)|$ and (G, k) is a yes-instance of Independent Set if and only if $(H, 2k)$ is a yes-instance of MVIKS problem. Thus this reduction can also be viewed as a reduction from Vertex Cover to KVDS problem. Dinur and Safra [5] have shown that unless P = NP, the Vertex Cover problem cannot be approximated to within 1.3606. Using this, one can easily show that the KVDS problem cannot be approximated to within 1.3606 unless P ≠ NP. The result of Dinur and Safra [5] also holds for graphs with a perfect matching. Using this fact, we show a stronger hardness result for the KVDS problem.

Corollary 4. *Under the hypothesis P ≠ NP, the KVDS problem cannot be approximated to within* 1.7212.

Proof. It is sufficient to prove this result for graphs with a perfect matching. Let $\mathcal{A}$ be a d-approximation algorithm for the KVDS problem in graphs with a perfect matching. Let G be an n-vertex graph with a perfect matching. Using $\mathcal{A}$, one can obtain a König vertex deletion set of size at most $d\kappa(G)$ and hence a vertex cover of size at most $\frac{n-d\kappa(G)}{2} + d\kappa(G) = \frac{n+d\kappa(G)}{2}$. An optimum vertex cover of G has size $\frac{n+\kappa(G)}{2}$. By Dinur and Safra [5], we must have $\frac{n+d\kappa(G)}{n+\kappa(G)} \geq 1.3606$. Simplifying this yields $\frac{n}{\kappa(G)} \leq \frac{d-1.3606}{0.3606}$. Note that $n \geq \kappa(G)$ and so $d \geq 1.7212$. ■

4 Edge Versions of the König Subgraph Problem

In this section we look at edge versions of König Subgraph problem.

4.1 NP-Completeness

We show that MEIKS problem is NP-complete by reducing the NP-complete Min 2-Sat Deletion problem to the KEDS problem.

Theorem 8. *The* König Edge Deletion Set (KEDS) *problem is NP-complete.*

Proof. We give a reduction from Min 2-Sat Deletion. Let Φ be a 2-Sat formula. Construct a graph $G_\Phi = (V, E)$ as follows. Suppose the formula Φ is composed of the literals $\{x_1, \bar{x}_1, \ldots, x_n, \bar{x}_n\}$. Let $V := \{x_1, \bar{x}_1, \ldots, x_n, \bar{x}_n, x_{11}, \bar{x}_{11}, \ldots, x_{1,k+1}, \bar{x}_{1,k+1}, \ldots, x_{n,1}, \bar{x}_{n,1}, \ldots, x_{n,k+1}, \bar{x}_{n,k+1}\}$; that is, V consists of $k + 2$ copies of $x_i, \bar{x}_i$. The edge set $E = E_1 \cup E_2 \cup E_3 \cup E_4$, where E_1, E_2, E_3, E_4 are defined below.

$$E_1 = \{(x_i, \bar{x}_i) : 1 \leq i \leq n\}$$
$$E_2 = \{(x_{i,j}, \bar{x}_{i,l}) : 1 \leq j, l \leq k+1, 1 \leq i \leq n\}$$
$$E_3 = \{(x_i, \bar{x}_{i,j}), (\bar{x}_i, x_{i,j}) : 1 \leq i \leq n, 1 \leq j \leq k+1\}$$
$$E_4 = \{(y_i, y_j) : (y_i \vee y_j) \in \Phi\}$$

Note that G_Φ has a perfect matching and that each clause of Φ corresponds to an edge of G_Φ (the edges in E_4).

Claim. There exists an assignment satisfying all but k clauses of Φ if and only if there exist at most k edges whose deletion makes G_Φ König.

($\Rightarrow$) Let α be an assignment to the variables of Φ that satisfies all but k clauses. Each of these k clauses corresponds to a distinct edge in G_Φ. Delete these edges from G_Φ. Then for each edge e in the remaining graph, at least one endpoint of e is assigned 1 by α. To prove that the remaining graph is König, by Lemma 1, we must demonstrate a bipartition of the vertex set into $V_1 \uplus V_2$ (say) such that V_2 is independent and there exists a matching saturating V_1 which crosses the cut (V_1, V_2). If $\alpha(x_i) = 1$ then place the vertices $x_i, x_{i,1}, \ldots x_{i,k+1}$ in V_1; else place $\bar{x}_i, \bar{x}_{i,1}, \ldots \bar{x}_{i,k+1}$ in V_1. The remaining vertices are placed in V_2. As Φ satisfies all remaining clauses, V_2 is independent. Note that if $x_i \in V_1$ then $\bar{x}_i \in V_2$ and vice versa. Also if $x_{i,j} \in V_1$ then $\bar{x}_{i,j} \in V_2$ and vice versa. Hence there exists a matching that saturates V_1 and crosses the cut (V_1, V_2).

($\Leftarrow$) Conversely suppose that deleting k edges makes G_Φ König. We will assume that this set of edges is a minimal edge deletion set. Call the resulting graph G'_Φ. Then the vertex set of G'_Φ can be partitioned into V_1 and V_2 such that V_2 is independent and there exists a matching saturating V_1 that crosses the cut (V_1, V_2).

Claim. For each $1 \leq i \leq n$, it is not the case that $x_i, \bar{x}_i \in V_1$ or $x_i, \bar{x}_i \in V_2$.

If both x_i and $\bar{x}_i$ are in V_1 then one can argue that there is no matching that saturates both x_i and $\bar{x}_i$. We skip the details. If both x_i and $\bar{x}_i$ are in V_2 then one can show that we end up deleting more than k edges.

Now it is easy to see that for each vertex y_i, all copies $y_{i,1}, \ldots, y_{i,k+1}$ of it must be placed in the same partition as y_i itself and hence all edges in E_1, E_2, E_3 lie across the cut (V_1, V_2). Therefore the edges that were deleted from G_Φ were from E_4. Each of these edges corresponds to a distinct clause in Φ. If a vertex y_i is in V_1 assign the corresponding literal the value 1; else assign the literal the value 0. Note that this assignment is consistent as all copies of a vertex are in the same partition as the vertex itself and for no variable do we have that $x_i, \bar{x}_i \in V_1$ or $x_i, \bar{x}_i \in V_2$. This assignment satisfies all but the k clauses that correspond to the edges that were deleted. ■

The parameterized complexity of KEDS is open.

4.2 Approximation Results

For the MEIKS problem, it is easy to obtain a 2-approximation algorithm by simply finding a cut of size $m/2$ and then deleting all the other edges. In this subsection, we give a 4/3-approximation algorithm for graphs with a perfect matching and a 5/3-approximation algorithm for general graphs based on following combinatorial lemmas.

Theorem 9. *Let $G = (V, E)$ be a graph with a maximum matching M and let $G_M = (V_M, E_M)$ be the graph induced on the vertices $V(M)$ of M. Then G has an edge-induced König subgraph of size at least $\frac{3|E_M|}{4} + \frac{|E-E_M|}{2} + \frac{|M|}{4}$.*

Proof. Randomly partition the vertex set of G into $V_1 \uplus V_2$ as follows. For each edge $e_i \in M$, select an endpoint of e_i with probability $1/2$ and place it in V_1. Define $V_2 = V - V_1$. Note that the edges in M always lie across the cut (V_1, V_2). An edge of $E_M - M$ is in $G[V_2]$ with probability $1/4$; an edge in $E - E_M$ lies in $G[V_2]$ with probability $1/2$. For each edge $e \in E$, define X_e to be the indicator random variable that takes the value 1 if $e \in G[V_2]$ and 0 otherwise. Also define $X = \sum_{e \in E} X_e$. Then $E[X] = \sum_{e \in E} E[X_e] = \frac{|E_M - M|}{4} + \frac{|E - E_M|}{2}$. Deleting the edges in $G[V_2]$ results in a König graph with $\frac{3|E_M|}{4} + \frac{|E - E_M|}{2} + \frac{|M|}{4}$ edges in expectation. This algorithm can be easily derandomized by the method of conditional probabilities. This completes the proof. ■

If $G = (V, E)$ has a perfect matching M then $E_M = E$ and $|M| = |V|/2$ and we have

Corollary 5. *Let $G = (V, E)$ be a graph on n vertices and m edges with a perfect matching. Then G has a subgraph with at least $\frac{3m}{4} + \frac{n}{8}$ edges that is König. This subgraph can be found in time $O(mn)$.*

Theorem 10. *Let $G = (V, E)$ be an undirected graph on n vertices and m edges. Then G has an edge-induced König subgraph of size at least $\frac{3m}{5}$.*

Proof. Let M be a maximum matching of G and let $G[V_M] = (V_M, E_M)$ be the subgraph induced by the vertices $V(M)$ of M. Let $\eta(G)$ denote the size of the maximum edge induced König subgraph of G. By Theorem 9, $\eta(G) \geq \frac{|E_M| + |M|}{4} + \frac{|E|}{2}$. Observe that by deleting all the edges in $G[V_M]$ we obtain a König subgraph of G. In fact, this is a bipartite graph with bipartition V_M and $V - V_M$. Therefore if $|E - E_M| \geq 3m/5$, the statement of the theorem clearly holds. Otherwise, $|E_M| \geq 2m/5$ and by Theorem 9, we obtain $\eta(G) \geq |M|/4 + 3m/5$. This completes the proof. ■

The following theorem follows from Corollary 5 and Theorem 10 and the fact that the optimum König subgraph has at most m edges.

Theorem 11. *The optimization version of the MEIKS problem is approximable within a factor of $5/3$ for general graphs. This factor can be improved to $4/3$ when restricted to graphs with a perfect matching.*

4.3 FPT Algorithms

Note that Theorem 10 actually shows that the parameterized version of the MKEIS problem is fixed-parameter tractable. To see this, suppose that (G, k) is an instance of the parameterized version of the MKEIS problem; we are to decide whether G has an edge induced König subgraph with at least k edges. Note that if the parameter $k \leq 3m/5$ then we answer YES and use the approximation algorithm described in the previous subsection to obtain an edge induced König subgraph with at least k edges. If $k > 3m/5$ then we simply use a trivial $O^*(2^m)$ brute-force algorithm to decide the question. This FPT algorithm has time complexity $O^*(2^{5k/3})$.

In this subsection, we give an $O^*(2^k)$ FPT algorithm for the MEIKS problem on connected graphs by using an exact algorithm for the optimization version of the problem.

To this end, we describe an $O^*(2^n)$ algorithm for this problem using a simple structural result characterizing minimal König edge deletion sets of a graph.

Theorem 12. *Let $G = (V, E)$ be a graph. If E' is a minimal König edge deletion set of G then there exists $V' \subseteq V$ such that $E(G[V']) = E'$, that is, the edge set of the subgraph induced by V' is precisely E'.*

Proof. Let E' be a minimal König edge deletion set of G. Then $G' = (V, E - E')$ is König. Then the vertex set of G' can be partitioned into V_1 and V_2 such that V_2 is a maximal independent set and there exists a matching saturating V_1 that lies across the cut (V_1, V_2). We claim that $V' = V_2$. Since E' is minimal, it is clear that $E(G[V_2]) = E'$. This completes the proof. ■

Our exact algorithm for the optimization version of MEIKS simply enumerates all possible subsets $V' \subseteq V$, deletes all edges E' in $G[V']$ and checks whether $G - E'$ is König. The algorithm returns an edge set $E' = E(G[V'])$ of smallest size such that $G - E'$ is König.

Theorem 13. *Given an n-vertex graph $G = (V, E)$, the optimization version of the* König Edge Deletion Set (KEDS) *(and hence the optimization version of MEIKS) can be solved in time $O^*(2^n)$ and space polynomial in n.*

Theorem 14. *The MEIKS problem can be solved in $O^*(2^k)$ time in connected undirected graphs.*

Proof. Let (G, k) be an instance of the MEIKS problem where G is a graph with m edges and n vertices. A connected graph has a spanning tree which, being bipartite, is König. Since a tree has $n - 1$ edges, if $k \leq n - 1$ we answer YES; else $n \leq k + 1$ and we use Theorem 13 to obtain an $O^*(2^k)$ time algorithm for the MEIKS problem. ■

5 Conclusion and Open Problems

In this paper, we introduced and studied vertex and edge versions of the König Subgraph problem from the points of view of parameterized complexity and approximation algorithms. There are several open problems specified in the list below. Important among them are whether the KVDS (g-VC) and KEDS problems are fixed parameter tractable.

Problem	*Parameterized Complexity*	*Approximability*
KVDS/g-VC	Open.	$O(\log n \log\log n)$ approximation algorithm; no PTAS.
KEDS	Open.	Open.
MVIKS	$W[1]$-hard.	no factor-$O(n^{1-\epsilon})$ approximation algorithm.
MEIKS	FPT.	5/3-approximation algorithm for general graphs; 4/3-approximation algorithm for graphs with a perfect matching

References

1. Agarwal, A., Charikar, M., Makarychev, K., Makarychev, Y.: $O(\sqrt{\log n})$ Approximation Algorithms for Min UnCut, Min-2CNF Deletion, and Directed Cut Problems. In: Proc. of STOC 2005, pp. 573–581 (2005)
2. Chen, J., Kanj, I.A.: On Approximating Minimum Vertex Cover for Graphs with Perfect Matching. Theoretical Computer Science 337, 305–318 (2005)
3. Chen, J., Kanj, I.A., Xia, G.: Improved Parameterized Upper Bounds for Vertex Cover. In: Královič, R., Urzyczyn, P. (eds.) MFCS 2006. LNCS, vol. 4162, pp. 238–249. Springer, Heidelberg (2006)
4. Deming, R.W.: Independence Numbers of Graphs – An Extension of the König-Egerváry Theorem. Discrete Mathematics 27, 23–33 (1979)
5. Dinur, I., Safra, S.: On the Hardness of Approximating Minimum Vertex Cover. Annals of Mathematics 162(1), 439–485 (2005)
6. Klein, P.N., Plotkin, S.A., Rao, S., Tardos, E.: Approximation Algorithms for Steiner and Directed Multicuts. Jour. of Algorithms 22(2), 241–269 (1997)
7. Korach, E., Nguyen, T., Peis, B.: Subgraph Characterization of Red/Blue-Split Graphs and König-Egerváry Graphs. In: the Proc. of SODA (2006)
8. Garey, M.R., Johnson, D.S.: Computers and Intractability. W. H. Freeman and Company, New York (1979)
9. Guo, J., Gramm, J., Hüffner, F., Niedermeier, R., Wernicke, S.: Compression-Based Fixed-Parameter Algorithms for Feedback Vertex Set and Edge Bipartization. Jour. of Comp. and Sys. Sc. 72(8), 1386–1396 (2006)
10. Håstad, J.: Clique is Hard to Approximate to within $n^{1-\epsilon}$. Acta Mathematica 182, 105–142 (1999)
11. Mahajan, M., Raman, V., Sikdar, S.: Parameterizing MAX SNP Problems Above Guaranteed Values. In: Bodlaender, H.L., Langston, M.A. (eds.) IWPEC 2006. LNCS, vol. 4169, pp. 38–49. Springer, Heidelberg (2006)
12. Niedermeier, R.: An Invitation to Fixed-Parameter Algorithms. Oxford University Press, Oxford (2006)
13. Reed, B., Smith, K., Vetta, A.: Finding Odd Cycle Transversals. Operations Research Letters 32, 299–301 (2004)
14. Schroder, H., May, A.E., Sykora, O., Vrt'o, I.: Algorithms for the Vertex Bipartization Problem. In: Jeffery, K.G. (ed.) SOFSEM 1997. LNCS, vol. 1338, pp. 547–554. Springer, Heidelberg (1997)
15. Subramanian, C.R.: Vertex Covers: Parameterizing Above the Requirement. IMSc Technical Report (February 2001)
16. Sterboul, F.: A Characterization of Graphs in which the Transversal Number Equals the Matching Number. Jour. of Comb. Theory, Series B 27, 228–229 (1979)
17. Vazirani, V.V.: A Theory of Alternating Paths and Blossoms for Proving Correctness of the $O(\sqrt{V}E)$ General Graph Maximum Matching Algorithm. Combinatorica 14(1), 71–109 (1994)
18. Zuckerman, D.: Linear Degree Extractors and the Inapproximability of Max Clique and Chromatic Number. In: the Proc. of STOC 2006, pp. 681–690 (2006)

New Bounds for the Nearly Equitable Edge Coloring Problem

Xuzhen Xie[1], Mutsunori Yagiura[1], Takao Ono[1], Tomio Hirata[1], and Uri Zwick[2]

[1] Graduate School of Information Science, Nagoya University, Nagoya 464-8603, Japan
sharryx@hirata.nuee.nagoya-u.ac.jp, yagiura@nagoya-u.jp
takao@hirata.nuee.nagoya-u.ac.jp, hirata@is.nagoya-u.ac.jp
[2] Department of Computer Science, Tel Aviv Univerusity, Tel Aviv 69978, Israel
zwick@tau.ac.il

Abstract. An edge coloring of a multigraph is nearly equitable if, among the edges incident to each vertex, the numbers of edges colored with any two colors differ by at most two. It has been proved that this problem can be solved in $O(m^2/k)$ time, where m and k are the numbers of edges and given colors, respectively. In this paper, we present a recursive algorithm that runs in $O\left(mn \log\left(m/(kn)+1\right)\right)$ time, where n is the number of vertices. This algorithm improves the best-known worst-case time complexity. When $k = O(1)$, the time complexity of all known algorithms is $O(m^2)$, which implies that this time complexity remains to be the best for more than twenty years since 1982 when Hilton and de Werra gave a constructive proof for the existence of a nearly equitable edge coloring for any graph. Our result is the first that improves this time complexity when m/n grows to infinity; e.g., $m = n^{\vartheta}$ for an arbitrary constant $\vartheta > 1$. We also propose a very simple randomized algorithm that runs in $O\left(m^{3/2}n^{1/2}/k^{1/2}\right)$ time with probability at least $1 - 1/c$ for any constant $c > 1$, whose worst-case time complexity is $O(m^2/k)$.

1 Introduction

Given a multigraph $G = (V, E)$ with n vertices and m edges and a color set $\mathcal{C} = \{1, 2, \ldots, k\}$, the nearly equitable edge coloring is an assignment of given colors to edges in G such that, among the edges incident to each vertex, the numbers of edges colored with any two colors differ by at most two. The notion of the nearly equitable edge coloring was introduced in 1982 by Hilton and de Werra [1] (a simplified version [2] was published in 1994), who also proved that any graph has a nearly equitable edge coloring. Their proof is constructive and easily leads to an algorithm for finding such a coloring in $O(km^2)$ time as mentioned in [3]. Later, Nakano et al. [3] showed an algorithm that runs

T. Tokuyama (Ed.): ISAAC 2007, LNCS 4835, pp. 280–291, 2007.

in $O(m^2/k + mn)$ time. In 2004, Xie et al. [4] presented a more efficient algorithm, which improved the running time to $O(m^2/k)$ and moreover satisfied the following *balanced constraint*: The numbers of the edges colored with any two colors differ by at most one. We call their algorithm BalCol in the following, which stands for *Balanced Coloring*.

In this paper, we present a recursive algorithm that runs in $O\left(mn \log\left(m/(kn)+1\right)\right)$ time, and a very simple randomized algorithm that runs in $O\left(m^{3/2}n^{1/2}/k^{1/2}\right)$ time with high probability and in $O(m^2/k)$ time in the worst case. Moreover, the edge colorings obtained by both of the two algorithms also satisfy the *balanced constraint*. We call the recursive algorithm RecCol, which stands for *Recursive Coloring*, and call the randomized algorithm RanCol, which stands for *random coloring*.

The time complexity of the recursive algorithm RecCol is better than the previous best known time bound $O(m^2/k)$ of algorithm BalCol. When $k = O(1)$, the time complexity of all known algorithms becomes $O(m^2)$, which implies that the time complexity, derived from the proof of Hilton and de Werra as referred, remains to be the best for more than twenty years. RecCol is the first algorithm that improves this time complexity for the case when $m/n \to \infty$ $(n \to \infty)$ holds; e.g., when $m = n^{\vartheta}$ ($\vartheta > 1$ is an arbitrary constant).

Both of RecCol and RanCol use a procedure that constitutes the core part of algorithm BalCol presented by Xie et al. [4]. We call this procedure ChkRec (*Check and Recolor*). Whenever the current edge coloring π is not nearly equitable, algorithm ChkRec invokes an algorithm called Recolor to modify π. Let $E_\pi(i)$ denote the set of edges colored with i. Xie et al. introduced a potential Φ_π and showed that Recolor runs in $O(|E_\pi(i) \cup E_\pi(j)|)$ time for relevant colors i and j and decreases Φ_π by at least one for each call to it. It is therefore preferable to construct a coloring π such that potential Φ_π is small and $|E_\pi(i)|$ is small for all $i \in \mathcal{C}$ before calling algorithm ChkRec to keep its computation time smaller.

When $|E| \leq kn$, algorithm RecCol uses algorithm BalCol directly to obtain a nearly equitable edge coloring π of G. Otherwise, it partitions E arbitrarily into two subsets E^{L} and E^{R} of about the same size, and applies RecCol to both of them to obtain edge colorings π^{L} and π^{R} of E^{L} and E^{R}, respectively. It then constructs a coloring π of G by merging the two colorings π^{L} and π^{R} after permuting them. We show that the resulting edge coloring π satisfies $|E_\pi(i)| = O(m/k)$ for all colors $i \in \mathcal{C}$ and $\Phi_\pi = O(kn)$.

Algorithm RanCol starts with repeatedly choosing $\lceil m/k \rceil$ or $\lfloor m/k \rfloor$ random edges and assigning them a new color until all the edges are colored. We show that such a coloring π satisfies $|E_\pi(i)| = O(m/k)$ for all colors $i \in \mathcal{C}$ and $\Phi_\pi = O\left((kmn)^{1/2}\right)$ with probability at least $1 - 1/c$ for any constant $c > 1$.

The rest of the paper is organized as follows. In Section 2, we give some definitions. Section 3 introduces the previous results presented by Xie et al., and Sections 4 and 5 explain the recursive algorithm and the randomized algorithm, respectively. Finally, concluding remarks are in Section 6.

2 Preliminaries

A multigraph G is a graph that allows multiple number of edges between vertices. Let V and E denote the sets of vertices and edges of G, respectively. The following definitions will be used throughout this paper.

- Let $n = |V|$ and $m = |E|$. Note that $m \geq n - 1$ holds for any connected graph.
- We denote the given color set by $\mathcal{C} = \{1, 2, \ldots, k\}$, where k is the number of given colors.
- We denote an edge coloring by a mapping π: $E \to \mathcal{C}$; i.e., if an edge $e \in E$ is colored with a color i, then $\pi(e) = i$.
- For each vertex $v \in V$, let $N(v) = \{(v, w) \in E\}$ denote the edges incident to v in G and $d(v) = |N(v)|$ be its degree. Then, $d_\pi(v, i) = |\{e \in N(v) \mid \pi(e) = i\}|$ stands for the number of edges colored with i and incident to v, while $E_\pi(i) = \{e \in E \mid \pi(e) = i\}$ stands for the set of edges in E colored with i.
- For any subset $E' \subseteq E$, let $V_{E'}$ be the set of end vertices of edges in E'; i.e., $V_{E'} = \{v \in V \mid \exists w \in V, (v, w) \in E'\}$. Then let $G_{E'} = (V_{E'}, E')$.
- Let $V_\pi(i, j)$ be the set of end vertices of edges in $E_\pi(i) \cup E_\pi(j)$; i.e., $V_\pi(i, j) = \{v \in V \mid \exists w \in V,\ (v, w) \in (E_\pi(i) \cup E_\pi(j))\}$. Then let $G_\pi(i, j) = (V_\pi(i, j), E_\pi(i) \cup E_\pi(j))$ be the subgraph whose edges are $E_\pi(i) \cup E_\pi(j)$ and vertices are their end vertices.

Then, the definitions of nearly equitable edge coloring introduced by Hilton and Werra [1], and the *balanced constraint* introduced by Nakano et al. [3] are as follows.

Definition 1 ([1]). *Given a multigraph $G = (V, E)$ and a color set $\mathcal{C} = \{1, 2, \ldots, k\}$, the nearly equitable edge coloring π is an assignment of the given k colors to all the edges in G, such that for any vertex $v \in V$ and different colors $i, j \in \mathcal{C}$, $|d_\pi(v, i) - d_\pi(v, j)| \leq 2$.*

Definition 2 ([3]). *An edge coloring π satisfies the balanced constraint if $||E_\pi(i)| - |E_\pi(j)|| \leq 1$ holds for any two colors i and j.*

Without loss of generality, we assume that G is connected, $|E| > 0$ and $k \geq 2$.

3 Previous Results for Nearly Equitable Edge Coloring

3.1 Previous Algorithm

In this section, we introduce a simplified version of algorithm BalCol, which is presented by Xie et al. [4]. Algorithm BalCol initially assigns k colors $1, 2, \ldots, k$ to k uncolored edges repeatedly until all the edges are colored and then calls algorithm ChkRec. Whenever the current coloring π has a vertex u that breaks the condition of Definition 1, algorithm ChkRec chooses two colors α and β with maximum and minimum $d_\pi(u, i)$, respectively, and calls algorithm Recolor to

recolor those edges in $E_\pi(\alpha) \cup E_\pi(\beta)$. Algorithm RECOLOR first constructs an augmented graph $\hat{G} = (\hat{V}, \hat{E})$ by adding a vertex $\hat{v}$ and some edges to make $G_\pi(\alpha, \beta)$ connected and the degrees of all vertices even. It then finds an Euler circuit in $\hat{G}$ starting at the additional vertex $\hat{v}$ and colors the edges alternately with α and β along the Euler circuit, so that the resulting coloring π' is balanced with respect to the two colors, i.e., $||E_{\pi'}(\alpha)| - |E_{\pi'}(\beta)|| \leq 1$ holds after recoloring.

Algorithm. BALCOL$(G, \mathcal{C})$
Input: a multigraph $G = (V, E)$ and a k-color set $\mathcal{C} = \{1, 2, \ldots, k\}$
Output: a nearly equitable edge coloring π for G

1. Generate an arbitrary permutation $\sigma : \{1, 2, \ldots, m\} \to E$, and then let $\pi : E \to \mathcal{C}$ be the edge coloring that satisfies $\pi(\sigma(l)) \equiv l \pmod{k}$.
2. Let $\pi \leftarrow$ CHKREC$(G, \mathcal{C}, \pi)$.
3. Output π and stop.

Algorithm. CHKREC$(G, \mathcal{C}, \pi)$
Input: a multigraph $G = (V, E)$, a k-color set $\mathcal{C} = \{1, 2, \ldots, k\}$, and an edge coloring π
Output: a nearly equitable edge coloring π for G

1. **while** there exists $u \in V$ and different colors $i, j \in \mathcal{C}$ such that $|d_\pi(u, i) - d_\pi(u, j)| \geq 3$ **do**
2. For the vertex u, find two colors $\alpha, \beta \in \mathcal{C}$ satisfying
   $d_\pi(u, \alpha) = \max_{i \in \mathcal{C}} (d_\pi(u, i))$
   $d_\pi(u, \beta) = \min_{i \in \mathcal{C}} (d_\pi(u, i))$.
3. Call RECOLOR(G, α, β, π).
4. Output π and stop.

Algorithm. RECOLOR(G, α, β, π)
Input: a multigraph $G = (V, E)$, colors α and β, and an edge coloring π
Task: modify the edge coloring π for $G_\pi(\alpha, \beta)$

1. Let $\hat{V} \leftarrow V_\pi(\alpha, \beta) \cup \{\hat{v}\}$ $(\hat{v} \notin V)$ and $\hat{E} \leftarrow E_\pi(\alpha) \cup E_\pi(\beta)$.
2. **for** each connected component H in $G_\pi(\alpha, \beta)$ **do**
3. **if** H has at least one odd-degree vertex **then**
4. For each odd-degree vertex v in H, add an edge $(v, \hat{v})$ into $\hat{E}$.
5. **else**
6. **if** H has a vertex v such that $|d_\pi(v, \alpha) - d_\pi(v, \beta)| \geq 2$ **then**
   Draw two parallel edges between v and $\hat{v}$,
   and add them into $\hat{E}$.
7. **else**
   Let v be an arbitrary vertex in H.
   Draw two parallel edges between v and $\hat{v}$,
   and add them into $\hat{E}$.

8. Let $\hat{G} \leftarrow (\hat{V}, \hat{E})$.
9. Let a sequence of edges $e_1, e_2, \ldots, e_l$ be an Euler circuit of $\hat{G}$ such that the tail of e_1 is $\hat{v}$. Then let $\hat{\pi}(e_t) \leftarrow \alpha$ if t is odd and $\hat{\pi}(e_t) \leftarrow \beta$ otherwise for all $t = 1, 2, \ldots, l$.
10. Let $\pi(e) \leftarrow \hat{\pi}(e)$ for all edges e in $G_\pi(\alpha, \beta)$, and stop.

3.2 Analysis of BalCol

It is obvious that when algorithm ChkRec stops, it outputs a nearly equitable edge coloring for the given multigraph. As the initial phase of edge coloring obviously uses $O(m)$ time, the running time of BalCol is dominated by the running time of ChkRec, which is decided by the running time of Recolor and the number of calls to Recolor. To analyze the number of calls to Recolor, Xie et al. introduced a potential Φ_π, which is defined as follows. For all vertices $v \in V$, let $\bar{d}(v) = \lfloor d(v)/k \rfloor$. Define

$$\Phi_\pi(v, i) = \varphi^2_{\bar{d}(v)-1}(d_\pi(v, i))$$
$$\Phi_\pi(v) = \sum_{i \in \mathcal{C}} \Phi_\pi(v, i)$$
$$\Phi_\pi = \sum_{v \in V} \Phi_\pi(v),$$

where $\varphi^2_{\bar{d}(v)-1}(d_\pi(v, i))$ is defined by

$$\varphi^b_a(x) = \max\{x - a - b, a - x, 0\} \tag{1}$$

with $x = d_\pi(v, i)$, $a = \bar{d}(v) - 1$ and $b = 2$. By definition, $\Phi_\pi \geq 0$ holds for any coloring π.

Below we summarize some results in [4], which are necessary for analyzing the time complexity of our algorithms RecCol and RanCol in the later sections.

Lemma 1 ([4]). *Let π and π' be the coloring before and after calling* Recolor. *Then* Recolor *runs in $O(|E_\pi(\alpha) \cup E_\pi(\beta)|)$ time,*[1] *and the coloring π' satisfies $||E_{\pi'}(\alpha)| - |E_{\pi'}(\beta)|| \leq 1$.*

Lemma 2 ([4]). *Assume that there exists a vertex u and colors α, β such that $d_\pi(u, \alpha) \geq \bar{d}(u) + 1$, $d_\pi(u, \beta) \leq \bar{d}(u)$ and $d_\pi(u, \alpha) - d_\pi(u, \beta) \geq 3$ for a coloring π, and let π' be the coloring after calling* Recolor. *Then $\Phi_{\pi'} \leq \Phi_\pi - 1$ holds.*

The initial phase of edge coloring π satisfies $||E_\pi(i)| - |E_\pi(j)|| \leq 1$ for any $i, j \in \mathcal{C}$, which implies that $|E_\pi(i)| = O(m/k)$ holds for all $i \in \mathcal{C}$. Lemma 1 implies that after invoking Recolor, the edge coloring π' also satisfies $||E_{\pi'}(i)| - |E_{\pi'}(j)|| \leq 1$ for any $i, j \in \mathcal{C}$; thus Recolor always runs in $O(m/k)$

[1] Note that algorithm Recolor modifies the coloring for $G_\pi(\alpha, \beta)$, which can be extracted from G in $O(|E_\pi(\alpha) \cup E_\pi(\beta)|)$ time if appropriately implemented.

time. Since $\Phi_\pi \geq 0$ holds for any π by definition, Lemma 2 implies that the number of calls to RECOLOR is bounded by the value of Φ_π for the initial coloring π, which can be calculated to be $O(m)$. Thus, algorithm BALCOL runs in $O(m^2/k)$ time and the following theorem holds.

Theorem 1 ([4]). *Algorithm* BALCOL *solves the nearly equitable edge coloring problem for multigraphs in $O(m^2/k)$ time, where m and k are the numbers of edges and given colors, respectively. Moreover, the edge coloring π satisfies the balanced constraint; i.e., $||E_\pi(i)| - |E_\pi(j)|| \leq 1$ for any two colors i, j, where $E_\pi(i)$ denotes the set of edges colored with i.*

4 A Recursive Algorithm for Nearly Equitable Edge Coloring

4.1 Recursive Algorithm

When $|E| \leq kn$, algorithm RECCOL calls the previous algorithm BALCOL to obtain an edge coloring of G. Otherwise, it partitions E arbitrarily into two subsets E^{L} and E^{R} so that they satisfy $E^{\mathrm{L}} \cup E^{\mathrm{R}} = E$, $E^{\mathrm{L}} \cap E^{\mathrm{R}} = \emptyset$, $|E^{\mathrm{L}}| = \lceil |E|/2 \rceil$ and $|E^{\mathrm{R}}| = \lfloor |E|/2 \rfloor$, and then applies RECCOL to $G_{E^{\mathrm{L}}}$ and $G_{E^{\mathrm{R}}}$ to obtain their edge colorings π^{L} and π^{R}, respectively. It then constructs a coloring π of G by merging the two colorings π^{L} and π^{R} after permuting them so that the resulting coloring π satisfies $||E_\pi(i)| - |E_\pi(j)|| \leq 1$ for any two colors $i, j \in \mathcal{C}$, and calls algorithm CHKREC.

Algorithm. RECCOL$(G, \mathcal{C})$
Input: a multigraph $G = (V, E)$ and a k-color set $\mathcal{C} = \{1, 2, \ldots, k\}$
Output: a nearly equitable edge coloring π for G

1. **If** $|E| \leq kn$ **then**
   Let $\pi \leftarrow$ BALCOL $(G, \mathcal{C})$.
2. **else**
3. Partition E into two subsets E^{L} and E^{R} so that they satisfy $E = E^{\mathrm{L}} \cup E^{\mathrm{R}}$, $E^{\mathrm{L}} \cap E^{\mathrm{R}} = \emptyset$, $|E^{\mathrm{L}}| = \lceil |E|/2 \rceil$ and $|E^{\mathrm{R}}| = \lfloor |E|/2 \rfloor$.
4. Let $\pi^{\mathrm{L}} \leftarrow$ RECCOL$(G_{E^{\mathrm{L}}}, \mathcal{C})$.
   Let $\pi^{\mathrm{R}} \leftarrow$ RECCOL$(G_{E^{\mathrm{R}}}, \mathcal{C})$.
5. Sort the color indices so that $\left|E^{\mathrm{L}}_{\pi^{\mathrm{L}}}(i_1)\right| \geq \left|E^{\mathrm{L}}_{\pi^{\mathrm{L}}}(i_2)\right| \geq \ldots \geq \left|E^{\mathrm{L}}_{\pi^{\mathrm{L}}}(i_k)\right|$ holds. Define a permutation $\sigma^{\mathrm{L}} : \mathcal{C} \to \mathcal{C}$ by $\sigma^{\mathrm{L}}(i_l) = l$ for all $l = 1, 2, \ldots, k$. Let $\pi(e) \leftarrow \sigma^{\mathrm{L}}\left(\pi^{\mathrm{L}}(e)\right)$ for all $e \in E^{\mathrm{L}}$.
6. Sort the color indices so that $\left|E^{\mathrm{R}}_{\pi^{\mathrm{R}}}(j_1)\right| \leq \left|E^{\mathrm{R}}_{\pi^{\mathrm{R}}}(j_2)\right| \leq \ldots \leq \left|E^{\mathrm{R}}_{\pi^{\mathrm{R}}}(j_k)\right|$ holds. Define a permutation $\sigma^{\mathrm{R}} : \mathcal{C} \to \mathcal{C}$ by $\sigma^{\mathrm{R}}(j_l) = l$ for all $l = 1, 2, \ldots, k$. Let $\pi(e) \leftarrow \sigma^{\mathrm{R}}\left(\pi^{\mathrm{R}}(e)\right)$ for all $e \in E^{\mathrm{R}}$.
7. Let $\pi \leftarrow$ CHKREC$(G, \mathcal{C}, \pi)$.
8. Return π.

4.2 Analysis of Algorithm RecCol

Since the edge coloring obtained by algorithm RecCol is the output of algorithm BalCol or ChkRec, it is obvious that algorithm RecCol outputs a nearly equitable edge coloring for the given multigraph when it stops. Let us consider the time complexity of each recursive call to algorithm RecCol (i.e., the computation time except Line 4): The edge coloring π given by algorithm BalCol of Line 1 uses $O(m^2/k)$ time by Theorem 1. The permutations in Lines 5 and 6 can be computed in $O(k) = O(m)$ time (recall that $|E| > kn$ holds when these lines are called), and hence the time complexities of Lines 3, 5 and 6 are $O(m)$. Thus the remaining is the time for algorithm ChkRec, which is determined by the running time of algorithm Recolor and the number of calls to Recolor. Lemmas 1 and 2 imply that the running time of Recolor is $O\left(\max_{i\in\mathcal{C}} |E_\pi(i)|\right)$ and the number of calls to Recolor is bounded by the value of Φ_π, where $E_\pi(i)$ and Φ_π are those of the coloring π just before calling ChkRec in Line 7. We first show the following lemmas.

Lemma 3. *The edge coloring π output by algorithm* RecCol *satisfies the balanced constraints; i.e., $||E_\pi(i)| - |E_\pi(j)|| \leq 1$ for any two colors $i, j \in \mathcal{C}$.*

Proof. We prove this by the induction on the number of edges.

When $|E| \leq kn$, we call algorithm BalCol, and in this case, Theorem 1 implies that the edge coloring π given by algorithm BalCol satisfies the balanced constraint. We now consider the case with $|E| > kn$, and assume that the lemma holds for any multigraph whose number of edges is smaller than $|E|$. Then, as $|E^{\mathrm{L}}|$ and $|E^{\mathrm{R}}|$ are smaller than $|E|$, after calling RecCol on both $G_{E^{\mathrm{L}}}$ and $G_{E^{\mathrm{R}}}$ in Line 4, the obtained two edge colorings π^{L} and π^{R} satisfy

$$\left|\left|E^{\mathrm{L}}_{\pi^{\mathrm{L}}}(i)\right| - \left|E^{\mathrm{L}}_{\pi^{\mathrm{L}}}(j)\right|\right| \leq 1 \tag{2}$$

$$\left|\left|E^{\mathrm{R}}_{\pi^{\mathrm{R}}}(i)\right| - \left|E^{\mathrm{R}}_{\pi^{\mathrm{R}}}(j)\right|\right| \leq 1 \tag{3}$$

for any two colors $i, j \in \mathcal{C}$ by the inductive hypothesis. Hence, after permuting the colors in Lines 5 and 6, we have

$$||E_\pi(i)| - |E_\pi(j)|| \leq 1 \tag{4}$$

for any two colors $i, j \in \mathcal{C}$, where π is the edge coloring just before calling ChkRec. Then, Lemma 1 implies that this condition still holds after calling algorithm ChkRec. □

Lemma 4. *The edge coloring π just before calling* ChkRec *in algorithm* RecCol *satisfies*

(i) $||E_\pi(i)| - |E_\pi(j)|| \leq 1$ *for all* $i, j \in \mathcal{C}$, *and*
(ii) $\Phi_\pi = O(kn)$.

Proof. (i) is obvious from the proof of Lemma 3. Below we give the proof of (ii).

To make the proof simple, we define $\hat{\Phi}_\pi(v,i)$ and $\hat{\Phi}_\pi$ as follows:

$$\begin{aligned}\hat{\Phi}_\pi(v,i) &= \varphi^0_{d(v)/k}\left(d_\pi(v,i)\right) = \left|d_\pi(v,i) - \frac{d(v)}{k}\right| \\ \hat{\Phi}_\pi(v) &= \sum_{i\in\mathcal{C}} \hat{\Phi}_\pi(v,i) \\ \hat{\Phi}_\pi &= \sum_{v\in V} \hat{\Phi}_\pi(v),\end{aligned} \tag{5}$$

where $\varphi^0_{d(v)/k}\left(d_\pi(v,i)\right)$ is defined by (1) with $x = d_\pi(v,i)$, $a = d(v)/k$ and $b = 0$. It is easy to show that for any constants a, a', $b \geq 0$ and $b' \geq 0$ satisfying $a \leq a'$ and $a' + b' \leq a + b$, $\varphi^b_a(x) \leq \varphi^{b'}_{a'}(x)$ holds for all x. Thus, for $\bar{d}(v) - 1 \leq d(v)/k$ and $(d(v)/k) + 0 \leq (\bar{d}(v) - 1) + 2$, we obtain $\Phi_\pi \leq \hat{\Phi}_\pi$ for all vertices $v \in V$ and all colors $i \in \mathcal{C}$.

Let $d^{\mathrm{L}}(v)$ denote the number of edges in E^{L} incident to the vertex v and $d^{\mathrm{L}}_{\pi^{\mathrm{L}}}(v,i)$ denote the number of edges in E^{L} that are incident to v and colored with i under the coloring π^{L}. The definitions of $d^{\mathrm{R}}(v)$ and $d^{\mathrm{R}}_{\pi^{\mathrm{R}}}(v,i)$ are similar. After calling RecCol$(G_{E^{\mathrm{L}}}, \mathcal{C})$, the obtained coloring π^{L} is nearly equitable for $G_{E^{\mathrm{L}}} = \left(V_{E^{\mathrm{L}}}, E^{\mathrm{L}}\right)$, and hence

$$\left|d^{\mathrm{L}}_{\pi^{\mathrm{L}}}(v,i) - d^{\mathrm{L}}_{\pi^{\mathrm{L}}}(v,j)\right| \leq 2 \tag{6}$$

holds for any vertex $v \in V_{E^{\mathrm{L}}}$ and any two colors $i, j \in \mathcal{C}$.

Since

$$\min_{i\in\mathcal{C}}\left(d^{\mathrm{L}}_{\pi^{\mathrm{L}}}(v,i)\right) \leq \frac{d^{\mathrm{L}}(v)}{k} \leq \max_{i\in\mathcal{C}}\left(d^{\mathrm{L}}_{\pi^{\mathrm{L}}}(v,i)\right),$$

we have

$$\left|d^{\mathrm{L}}_{\pi^{\mathrm{L}}}(v,i) - \frac{d^{\mathrm{L}}(v)}{k}\right| \leq 2 \tag{7}$$

for any vertex $v \in V_{E^{\mathrm{L}}}$ and any color $i \in \mathcal{C}$. For the same reason, the coloring π^{R} obtained by calling RecCol$(G_{E^{\mathrm{R}}}, \mathcal{C})$ satisfies

$$\left|d^{\mathrm{R}}_{\pi^{\mathrm{R}}}(v,i) - \frac{d^{\mathrm{R}}(v)}{k}\right| \leq 2, \tag{8}$$

for any vertex $v \in V_{E^{\mathrm{R}}}$ and any color $i \in \mathcal{C}$. Since $d_\pi(v,i) = d^{\mathrm{L}}_{\pi^{\mathrm{L}}}\left(v, \left(\sigma^{\mathrm{L}}\right)^{-1}(i)\right) + d^{\mathrm{R}}_{\pi^{\mathrm{R}}}\left(v, \left(\sigma^{\mathrm{R}}\right)^{-1}(i)\right)$ and $d(v) = d^{\mathrm{L}}(v) + d^{\mathrm{R}}(v)$ hold by definition, (7) and (8) imply that the coloring π before calling ChkRec satisfies

$$\left|d_\pi(v,i) - \frac{d(v)}{k}\right| \leq 4. \tag{9}$$

Hence we have $\Phi_\pi \leq \hat{\Phi}_\pi \leq 4kn$. □

4.3 Running Time of Algorithm RecCol

We now consider the running time of algorithm RecCol. As discussed in Section 4.2, the running time of each recursive call to RecCol is dominated by the computation time of ChkRec in Line 7, which is determined by the running time of Recolor and the number of calls to Recolor. Lemma 4-(i) and Lemma 1 imply that Recolor always runs in $O\left(\max_{i\in\mathcal{C}} |E_\pi(i)|\right) = O(m/k)$ time, while Lemma 4-(ii) and Lemma 2 imply that the number of calls to Recolor is bounded by $\Phi_\pi = O(kn)$. Thus, the running time of ChkRec becomes $O(kn \times (m/k)) = O(mn)$.

Let $T(m,n,k)$ denote the total running time of algorithm RecCol. Then we have

$$T(m,n,k) = \begin{cases} O(m^2/k) & \text{if } m \le kn \\ O(mn) + 2T(m/2,n,k) & \text{otherwise.} \end{cases}$$

Since $m^2/k = O(mn)$ holds for $m \le kn$, we have $T(m,n,k) = O\left(mn \log\left(m/(kn)+1\right)\right)$ for $m > kn$. For $m \le kn$, $T(m,n,k) = O(m^2/k)$ is obvious. Below we show that $m^2/k = \Theta\left(mn \log\left(m/(kn)+1\right)\right)$ holds for $m \le kn$, which implies that $T(m,n,k) = O(mn \log(m/(kn)+1))$ holds for all range of m. We also show that $mn \log(m/(kn)+1) = O(m^2/k)$, which means that algorithm RecCol is never slower than BalCol. For these, we use the following lemma.

Lemma 5. *$x \le \lg(x+1)$ holds for any $x \in [0,1]$, while $\lg(x+1) \le 2x$ holds for any $x \ge 0$, where $\lg = \log_2$.*

Proof. Considering the function

$$f_c(x) = cx - \lg(x+1)$$

for $x \ge 0$ and a constant c, we have

$$f_c'(x) = c - \frac{1}{(x+1)\ln 2}$$
$$f_c''(x) = \frac{1}{(x+1)^2 \ln 2} > 0. \tag{10}$$

For $c = 1$, since $f_1(0) = f_1(1) = 0$ holds and (10) implies that $f_1(x)$ is a convex function, we have $f_1(x) \le 0$ for $x \in [0,1]$, which implies that $x \le \lg(x+1)$ holds for any $x \in [0,1]$. For $c = 2$, we have $f_2'(0) = 2 - 1/\ln 2 = 0.5573\cdots > 0$ and (10) implies that $f_2'(x)$ is monotonically increasing with x, $f_2'(x) > 0$ holds for any $x \ge 0$, which implies that $f_2(x)$ is monotonically increasing with x for $x \ge 0$. Then, by $f_2(0) = 0$, we have $f_2(x) \ge 0$ for any $x \ge 0$, which implies that $\lg(x+1) \le 2x$ holds for any $x \ge 0$. □

Lemma 5 implies that

$$mn \lg\left(\frac{m}{kn} + 1\right) \ge mn \frac{m}{kn} = \frac{m^2}{k}$$

holds for $0 < m/(kn) \leq 1$, while

$$mn \lg \left(\frac{m}{kn} + 1 \right) \leq mn \frac{2m}{kn} = \frac{2m^2}{k}$$

holds for any m, n and k. Thus, we conclude that $T(m, n, k) = O(mn \log(m/(kn) + 1))$ always holds, and algorithm RecCol is never asymptotically slower than algorithm BalCol. The results are summarized in the following theorem.

Theorem 2. *Algorithm* RecCol *solves the nearly equitable edge coloring problem for multigraphs in* $O(mn(\log(m/(kn)) + 1))$ *time, where* m, n *and* k *are the numbers of edges, vertices and given colors, respectively. Moreover, the edge coloring* π *satisfies the balanced constraint; i.e.,* $||E_\pi(i)| - |E_\pi(j)|| \leq 1$ *holds for any two colors* $i, j \in \mathcal{C}$.

When $k = O(1)$, the time complexity of all known algorithms becomes $O(m^2)$, which implies that the time complexity, derived from the constructive proof of Hilton and de Werra [1], remains to be the best. We now consider the ratio

$$\frac{mn \log(m/n)}{m^2} = \frac{\log(m/n)}{m/n}.$$

Then we have

$$\lim_{m/n \to \infty} \frac{\log(m/n)}{m/n} = 0,$$

which implies that algorithm RecCol needs less running time than $O(m^2)$ when m/n grows to infinity; e.g., $m = n^\vartheta$ ($\vartheta > 1$ is an arbitrary constant).

5 A Randomized Algorithm for Nearly Equitable Edge Coloring

5.1 Randomized Algorithm

Algorithm RanCol starts by repeatedly choosing $\lceil m/k \rceil$ or $\lfloor m/k \rfloor$ random edges and giving them a new color until all the edges are colored. This is equivalent to the following rule: We first generate a random permutation $\sigma : \{1, 2, \ldots, m\} \to E$ of edges, and then give the edges colors $1, 2, \ldots, k$ in this order in a circular manner (i.e., the next color of k is 1) according the order of σ. Algorithm RanCol then calls algorithm ChkRec.

Algorithm. RanCol$(G, \mathcal{C})$
Input: a multigraph $G = (V, E)$ and a k-color set $\mathcal{C} = \{1, 2, \ldots, k\}$
Output: a nearly equitable edge coloring π for G

1. Generate a random permutation $\sigma : \{1, 2, \ldots, m\} \to E$
   (each σ is generated with probability $1/m!$),
   and let $\pi : E \to \mathcal{C}$ be the coloring that satisfies $\pi(\sigma(l)) \equiv l \pmod{k}$.
2. Let $\pi \leftarrow$ ChkRec$(G, \mathcal{C}, \pi)$.
3. Output π and stop.

5.2 Time Complexity of Algorithm RanCol

As the initial phase of coloring randomly requires $O(m)$ time, the running time of algorithm RanCol is decided by the running time of Recolor and the number of calls to Recolor. For the random coloring π, $||E_\pi(i)| - |E_\pi(j)|| \leq 1$ is automatically satisfied for any two colors $i, j \in \mathcal{C}$, which implies that $|E_\pi(i)| = O(m/k)$ holds for all colors $i \in \mathcal{C}$ and hence Recolor runs in $O(m/k)$ time by Lemma 1. To analyze the number of calls to Recolor, we evaluate the values of Φ_π and $\hat{\Phi}_\pi$ with respect to the random coloring π. We summarize the results in the following lemma, whose proof is elaborate and long, and is omitted due to space limitation.

Lemma 6. *Given an n-vertex m-edge multigraph $G = (V, E)$ and a k-color set $\mathcal{C} = \{1, 2, \ldots, k\}$, use the way in Line 1 of algorithm* RanCol *to randomly choose $\lceil m/k \rceil$ or $\lfloor m/k \rfloor$ edges and assign them a new color until all the edges are colored. Then, such a random coloring π satisfies $|E_\pi(i)| = O(m/k)$ automatically for all colors $i \in \mathcal{C}$ and $\Phi_\pi \leq c(14kmn)^{1/2}$ with probability at least $1 - 1/c$ for any constant $c > 1$.*

We now consider the total running time of algorithm RanCol. By Lemma 6, $|E_\pi(i)| = O(m/k)$ for all colors $i \in \mathcal{C}$ and $\Phi_\pi = O\left((kmn)^{1/2}\right)$ holds with probability at least $1 - 1/c$ after the initial random coloring. When this happens, recalling that $\Phi_\pi \geq 0$, Lemmas 1 and 2 imply that the running time of algorithm RanCol is

$$O\left((kmn)^{1/2}\left(\frac{m}{k}\right)\right) = O\left(\frac{m^{3/2}n^{1/2}}{k^{1/2}}\right). \tag{11}$$

It is also clear that its worst-case time complexity is the same as that of BalCol. Consequently, we obtain the following theorem.

Theorem 3. *Algorithm* RanCol *solves the nearly equitable edge coloring problem for multigraphs in $O\left(m^{3/2}n^{1/2}/k^{1/2}\right)$ time with probability at least $1 - 1/c$ for any constant $c > 1$, and in $O(m^2/k)$ time in the worst case, where m, n and k are the numbers of edges, vertices and given colors, respectively. Moreover, the edge coloring π satisfies the balanced constraint; i.e., $||E_\pi(i)| - |E_\pi(j)|| \leq 1$ holds for any two colors $i, j \in \mathcal{C}$.*

Though the time complexities of deterministic and randomized algorithms cannot be compared directly, below we consider when $m^{3/2}n^{1/2}/k^{1/2}$ becomes smaller than m^2/k and $mn \lg(m/(kn) + 1)$, the formulae giving the worst-case time complexities of algorithms BalCol and RecCol, respectively, to observe when it is beneficial to use the randomized algorithm RanCol.

The condition

$$\frac{m^{3/2}n^{1/2}}{k^{1/2}} < \frac{m^2}{k}$$

is equivalent to $m/(kn) > 1$. In such a case, RanCol runs faster than BalCol with high probability.

It is not hard to show that

$$\frac{mn\lg(m/(kn)+1)}{m^{3/2}n^{1/2}/k^{1/2}} = \frac{\lg(m/(kn)+1)}{(m/(kn))^{1/2}} = O(1)$$

holds. Thus, regarding the time complexity, RanCol has no merit over RecCol; however, RanCol is still useful for its simplicity.

6 Concluding Remarks

In this paper, we presented two algorithms to compute nearly equitable edge colorings for multigraphs. The recursive algorithm runs in $O\left(mn\log\left(m/(kn)+1\right)\right)$ time, where m, n and k are the numbers of edges, vertices and given colors, respectively. The time complexity of the recursive algorithm is better than the previous best known time bound $O(m^2/k)$ presented by Xie et al. When $k = O(1)$, the time complexity of all known algorithms becomes $O(m^2)$, which implies that this time complexity remains to be the best for more than twenty years since 1982 when Hilton and de Werra gave a constructive proof for the existence of a nearly equitable edge coloring for any multigraph. Our recursive algorithm is the first that improves this time complexity when m/n grows to infinity; e.g., $m = n^{\vartheta}$ ($\vartheta > 1$ is an arbitrary constant). The randomized algorithm is very simple, and it runs in $O\left(m^{3/2}n^{1/2}/k^{1/2}\right)$ time with a constant probability arbitrarily close to 1 and in $O(m^2/k)$ in the worst case.

Acknowlegements. We would like to thank the anonymous referees for valuable comments. This research was supported in part by Grant-in-Aid for Scientific Research from the Ministry of Education, Culture, Sports, Science and Technology of Japan.

References

1. Hilton, A.J.W., de Werra, D.: Sufficient conditions for balanced and for equitable edge-colouring of graphs. O. R. Working paper 82/3, Dépt. of Math. École Polytechnique Fédérate de Lausanne, Switzerland (1982)
2. Hilton, A.J.W., de Werra, D.: A sufficient condition for equitable edge-colourings of simple graphs. Discrete Mathematics 128, 179–201 (1994)
3. Nakano, S., Suzuki, Y., Nishizeki, T.: An algorithm for the nearly equitable edge-coloring of graphs (in Japanese). The IEICE Transactions on Information and Systems (Japanese Edition) J78-D-I, 437–444 (1995)
4. Xie, X., Ono, T., Nakano, S., Hirata, T.: An improved algoirthm for the nearly equitable edge-coloring problem. IEICE Transactions on Fundamentals E87-A, 1029–1033 (2004)

Approximation to the Minimum Cost Edge Installation Problem

Ehab Morsy and Hiroshi Nagamochi

Department of Applied Mathematics and Physics
Graduate School of Informatics
Kyoto University
Yoshida Honmachi, Sakyo, Kyoto 606-8501, Japan
{ehab,nag}@amp.i.kyoto-u.ac.jp

Abstract. We consider the *minimum cost edge installation problem* (MCEI) in a graph $G = (V, E)$ with edge weight $w(e) \geq 0$, $e \in E$. We are given a vertex $s \in V$ designated as a sink, an edge capacity $\lambda > 0$, and a source set $S \subseteq V$ with demand $q(v) \in [0, \lambda]$, $v \in S$. For any edge $e \in E$, we are allowed to install an integer number $h(e)$ of copies of e. The MCEI asks to send demand $q(v)$ from each source $v \in S$ along a single path P_v to the sink s. A set of such paths can pass through a single copy of an edge in G as long as the total demand along the paths does not exceed the edge capacity λ. The objective is to find a set $\mathcal{P} = \{P_v \mid v \in S\}$ of paths of G that minimizes the installing cost $\sum_{e \in E} h(e)w(e)$. In this paper, we propose a $(15/8 + \rho_{\mathrm{ST}})$-approximation algorithm to the MCEI, where ρ_{ST} is any approximation ratio achievable for the *Steiner tree problem*.

Keywords: Approximation algorithm, Graph algorithm, Routing problem, Network optimization.

1 Introduction

We study a problem of finding routings from a set of sources to a single sink in a network with an edge installing cost. This problem is a fundamental and economically significant one that arises in hierarchical design of telecommunication networks [2] and transportation networks [8,9]. In telecommunication networks this corresponds to installing transmission facilities such as fiber-optic cables, which represent the edges of the network. In other applications, optical cables may be replaced by pipes, trucks, and so on.

Consider an edge-weighted undirected graph G, where $V(G)$ and $E(G)$ denote the vertex set and edge set of G, respectively. We are given a set $S \subseteq V(G)$ of vertices specified as sources and a vertex $s \in V(G)$ specified as a sink. Each source $v \in S$ has a nonnegative demand $q(v)$, all of which must be routed to s through a single path. We are also given a finite set of different cable types, where each cable type is specified by its capacity and its cost per unit weight. The costs of cables obey economies of scale, i.e., the cost per unit capacity per unit weight of a high capacity cable is significantly less than that of a low capacity cable.

T. Tokuyama (Ed.): ISAAC 2007, LNCS 4835, pp. 292–303, 2007.

The *single-sink buy-at-bulk problem* (SSBB) (also known as the *single-sink edge installation problem* [3]) asks to construct a network of cables in the graph by installing an integer number of each cable type between adjacent vertices in G so that given demands at the sources can be routed simultaneously to s. The goal is to minimize the costs of installed cables.

The problem of buy-at-bulk network design was first introduced by Salman et al. [8]. They showed that the problem is NP-hard by showing a reduction from the *Steiner tree problem*. The Steiner tree problem is a classical NP-hard optimization problem, and the current best approximation ratio for the Steiner tree problem is a bit less than 1.55 [7]. Moreover, they showed that the problem remains NP-hard even when only one cable type is available. The approximation ratio for the SSBB problem was gradually reduced from $O(\log^2 n)$ [1] to 24.92 [2] by a series of papers, where n is the number of vertices of the underlying graph.

In this paper, we study a special case of the SSBB that arises from transportation networks [9]. A multinational corporation wishes to enter a new geographic area, characterized by demand at each city. It has identified the location of its manufacturing facility. Suppose the shipping of the good will be carried out by some transport company. This transport company has only one truck type, with a fixed capacity. For each truck, the transport company charges at a fixed rate per mile, and offers no discount in the case where the truck is not utilized to full capacity. The problem facing the corporation is to decide a shipping plan of the finished good to each city, so that the total demand at each city is met and the total cost is minimized.

In such a transportation network, we have a single cable type with a fixed capacity $\lambda > 0$ for all edges, and we are interested in constructing a set $\mathcal{P}$ of paths each of which connects one of given sources to a single sink s. The cost of installing a copy of an edge e is represented by the weight of e. A subset of paths of $\mathcal{P}$ can pass through a single copy of an edge e as long as the total demand of these paths does not exceed the edge capacity λ; any integer number of separated copies of e are allowed to be installed. However, the demand of each source is not allowed to be split at any vertex or over two or more copies of the same edge. The cost of a set $\mathcal{P}$ of paths is defined by the minimum cost of installing copies of edges such that the demand of each source can be routed to the sink under the edge capacity constraint, i.e.,

$$cost(\mathcal{P}) = \sum_{e \in E(G)} h(e)w(e),$$

where $h(e)$ is the minimum number of copies of e required for routing the set of all demands along e, simultaneously. The goal is to find a set $\mathcal{P}$ of paths that minimizes $cost(\mathcal{P})$. We call this problem, the *minimum cost edge installation problem* (MCEI). Notice that, in order to get a feasible solution to the MCEI, such edge capacity λ should be as much as the maximum demand in the network. The MCEI can be formally defined as follows, where R^+ denotes the set of nonnegative reals.

Minimum Cost Edge Installation Problem (MCEI)
Input: A connected graph G, an edge weight function $w : E(G) \to R^+$, a sink $s \in V(G)$, a set $S \subseteq V(G)$ of sources, an edge capacity $\lambda > 0$, and a demand function $q : S \to R^+$ such that $q(v) \leq \lambda$, $v \in S$.
Feasible solution: A set $\mathcal{P} = \{P_v \mid v \in S, \{s, v\} \subseteq V(P_v)\}$ of paths of G.
Goal: Find a feasible solution $\mathcal{P}$ that minimizes $cost(\mathcal{P})$.

The MCEI is closely related to the *capacitated network design problem* (CND), which can be stated as follows. We are given an undirected graph G such that each edge $e \in E(G)$ is weighted by nonnegative real $w(e)$, a subset $S \subseteq V(G)$ of sources, and a vertex $s \in V(G)$ designated as a sink. Each source $v \in S$ has a nonnegative demand $q(v)$, all of which must be routed to s through a single path. A cable with fixed capacity λ is available for installing on the edges of the graph, where installing i copies of the cable on edge e costs $iw(e)$ and provides $i\lambda$ capacity. The CND asks to find a minimum cost installation of cables that provides sufficient capacity to route all of the demand simultaneously to s. The problem requires choosing a path from each source to the sink and finding the number of cables to be installed on each edge such that all the demand is routed without exceeding edge capacities. Demands of different sources may share the capacity on the installed cables and the capacity installed on an edge has to be at least as much as the total demand routed through this edge. For this problem, Mansour and Peleg [6] gave an $O(\log n)$-approximation algorithm for a graph with n vertices. Salman et al. [8] designed a 7-approximation algorithm for the CND based on a construction from [5]. Afterwards Hassin et al. [4] gave a $(2+\rho_{\text{ST}})$-approximation algorithm, where ρ_{ST} is any approximation ratio achievable for the Steiner tree problem. By using of a slight intricate version of this algorithm, they improved the approximation ratio to $(1+\rho_{\text{ST}})$ when every source has unit demand. When all non-sink vertices are sources, the approximation ratio of Hassin et al. [4] becomes 3 for general demands and 2 for unit demands, since the Steiner tree problem in this case is a minimum spanning tree problem.

Note that, a solution to each of the MCEI and the CND can be characterized by specifying for each source v, the path P_v along which the demand $q(v)$ of v will be sent to the sink. The cables installed on each edge of the network are induced by these paths. In particular, for each edge e, a feasible solution to the MCEI assigns an integer number of separated cable copies required for routing all demands in $\{q(v) \mid e \in E(P_v)\}$, simultaneously. On the other hand, a feasible solution to the CND assigns on e at least $\lceil \sum_{v:e\in E(P_v)} q(v)/\lambda \rceil$ copies of the cable. That is, on contrary to the MCEI, the CND allows the demand from a source to be split among different copies of the same edge. Note that, the algorithm of Hassin et al. [4] to the CND takes the advantage (over the MCEI) of this assumption only for routing demands larger than λ to the sink. Hence, their algorithms to the CND can be used to obtain approximate solutions to the MCEI with approximation ratios $1 + \rho_{\text{ST}}$ and $2 + \rho_{\text{ST}}$ for the unit and general demand networks, respectively. In this paper, we proved that there is a $(15/8 + \rho_{\text{ST}})$-approximation algorithm to the MCEI with general demands. Our

result is based on a new and elaborated method for partitioning the source set of a given tree. When $S = V(G)$, the approximation ratio becomes 2.875.

The rest of this paper is organized as follows. Section 2 introduces terminologies on graphs and two lower bounds on the optimal value of the MCEI. Section 3 describes some results on tree partitions. Section 4 gives a framework of our approximation algorithm for the MCEI, analyzing its approximation ratio. Section 5 makes some concluding remarks.

2 Preliminaries

This section introduces some notations and definitions. Let G be a simple undirected graph. We denote by $V(G)$ and $E(G)$ the sets of vertices and edges in G, respectively. An edge-weighted graph is a pair (G, w) of a graph G and a nonnegative weight function $w : E(G) \to R^+$. The length of a shortest path between two vertices u and v in (G, w) is denoted by $d_{(G,w)}(u, v)$. Given a vertex weight function $q : V(G) \to R^+$ in G, we denote by $q(Z)$ the sum $\sum_{v \in Z} q(v)$ of weights of all vertices in a subset $Z \subseteq V(G)$.

Let T be a tree. A *subtree* of T is a connected subgraph of T. For a subset $X \subseteq V(T)$ of vertices, let $T\langle X\rangle$ denote the minimal subtree of T that contains X (note that $T\langle X\rangle$ is uniquely determined). Now let T be a rooted tree. We denote by $L(T)$ the set of leaves in T. For a vertex v in T, let $Ch(v)$ and $D(v)$ denote the sets of children and descendants of v, respectively, where $D(v)$ includes v. A *subtree* T_v *rooted* at a vertex v is the subtree induced by $D(v)$, i.e., $T_v = T\langle D(v)\rangle$. For an edge $e = (u, v)$ in a rooted tree T, where $u \in Ch(v)$, the subtree induced by $\{v\} \cup D(u)$ is denoted by T_e, and is called a *branch* of T_v. For a rooted tree T_v, the *depth* of a vertex u in T_v is the length (the number of edges) of the path from v to u.

For a set Z, a set $\{Z_1, Z_2, ..., Z_\ell\}$ of pairwise disjoint subsets of Z is called a *partition* of Z if $\cup_{i=1}^{\ell} Z_i = Z$.

The rest of this section presents two lower bounds on the optimal value to the MCEI. The first lower bound has been proved and used to derive approximation algorithms to the CND in [4].

Lemma 1. *For an instance $I = (G = (V, E), w, S, q, s, \lambda)$ of the MCEI, let $opt(I)$ be the weight of an optimal solution to I, and T^* be the minimum weight of a tree that spans $S \cup \{s\}$ in G. Then*

$$\max\left\{ w(T^*), \frac{1}{\lambda} \sum_{t \in S} q(t) d_{(G,w)}(s, t) \right\} \leq opt(I),$$

where $w(T^*)$ is the sum of weights of edges in T^*. □

The second lower bound is derived from an observation on the distance from sources $t \in S$ with $q(t) > \lambda/2$ to sink s.

Lemma 2. *For an instance $I = (G = (V, E), w, S, q, s, \lambda)$ of the MCEI with $q(t) \in [0, \lambda]$, $t \in S$, let $opt(I)$ be the weight of an optimal solution to I. Then*

$$\sum_{t \in S'} d_{(G,w)}(s, t) \leq opt(I),$$

where $S' = \{t \in S \mid q(t) > \lambda/2\}$.

Proof. The proof is followed directly by noting that for any two sources $u, v \in S'$, the paths P_v and P_u of the optimal solution cannot share the capacity of a single copy of any edge $e \in E$. □

Given an instance $I = (G = (V, E), w, S, q, s, \lambda)$ of the MCEI, our algorithm firstly produces a tree T of G that spans all vertices in $S \cup \{s\}$, finds a partition $\mathcal{S}$ of S, and assigns a vertex $t_Z \in Z$ for each subset $Z \in \mathcal{S}$ such that when all demands in each subset $Z \in \mathcal{S}$ are routed to t_Z simultaneously, the total flow on each edge of T is at most λ, where we call such a vertex t_Z the *hub vertex* of Z. Afterward, for each $Z \in \mathcal{S}$, we install a copy of each edge in a shortest path $SP(s, t_Z)$ between s and t_Z in G, and construct path P_t, $t \in Z$, by adding $SP(s, t_Z)$ to the path between t and t_Z in T. The running time of this algorithm is dominated by the approximation algorithm for the Steiner tree problem to compute tree T.

3 Tree Partition

The purpose of this section is to describe how to construct a tree partition in a tree that spans a source set, that is, how to find a partition of the source set of the tree. Such a tree partition will be the basis of our approximation algorithm to the MCEI in the next section. We first present some results for special cases of tree partitioning.

3.1 Tree Partition in Special Trees

In this subsection, we prepare several lemmas on tree partition problem for a tree with special structure. We first introduce a subgraph which plays a key role in our algorithm.

Definition 1. *For a vertex v in a rooted tree, a source set $Z_v \subseteq V(T_v) - \{v\}$, a demand function $q : Z_v \to R^+$, and a positive number λ, a binary rooted tree T_v is said to be a* balance-tree *if $q(Z_v) > \lambda$ holds and the total demand in each of its branches is less than $(4/7)\lambda$.*

We are given a binary rooted tree T_x with a source set $Z_x = L(T_x)$, an edge capacity $\lambda > 0$, a demand function $q : Z_x \to R^+$ such that $q(t) \leq \lambda/2$ for all $t \in Z_x$, and a vertex weight function $d : Z_x \to R^+$. Moreover, for each $u \in Ch(x)$, if $q(V(T_u) \cap Z_x) \geq (4/7)\lambda$, then T_u contains a balance-tree and satisfies $q(V(T_u) \cap Z_x) < (8/7)\lambda$. We partition Z_x into subsets, and choose a hub

vertex from each subset such that, when demands of each subset are routed to its hub vertex simultaneously, the total flow on each edge of T_x is bounded from above by λ. For brevity, we use $(T_x, Z_x, q, d, \lambda)$ to refer to this tree throughout this subsection.

We present the following three lemmas without proofs due to space limitation.

Lemma 3. *Given a tree $(T_x, Z_x, q, d, \lambda)$ with $(8/7)\lambda \leq q(Z_x) < (12/7)\lambda$, there is a partition $\{X, Y\}$ of Z_x such that $q(Y) \geq (4/7)\lambda$, and when $q(X)$ and $q(Y)$ are routed to $t_X = \mathrm{argmin}\{d(t) \mid t \in Z_x\}$ and $t_Y = \mathrm{argmin}\{d(t) \mid t \in Y\}$, respectively, the total amount of these flow on each edge of T_x is at most λ.* □

Lemma 4. *Given a tree $(T_x, Z_x, q, d, \lambda)$ with $q(Z_x) \geq (12/7)\lambda$, there is a partition $\{A, B, C\}$ of Z_x and a subset $Z'_x \subseteq Z_x$ with $q(Z'_x) \geq (12/7)\lambda$ such that $q(A \cap Z'_x), q(B \cap Z'_x) > (3/7)\lambda$, $q(C \cap Z'_x) \geq (5/7)\lambda$, and when $q(A)$, $q(B)$, and $q(C)$ are routed to $t_A = \mathrm{argmin}\{d(t) \mid t \in Z'_x\}$, $t_B = \mathrm{argmin}\{d(t) \mid t \in Z'_x - A\}$, and $t_C = \mathrm{argmin}\{d(t) \mid t \in C \cap Z'_x\}$, respectively, the total amount of these flow on each edge of T_x is at most λ.* □

Lemma 5. *Given a tree $(T_x, Z_x, q, d, \lambda)$ with $Z_x \neq \emptyset$, there is a partition $\mathcal{Z}_1 \cup \mathcal{Z}_2$ of Z_x such that $q(Z) \geq (5/7)\lambda$ for each $Z \in \mathcal{Z}_1$, $q(Z) < (4/7)\lambda$ for each $Z \in \mathcal{Z}_2$, and when demands in each $Z \in \mathcal{Z}_1$ and $Z \in \mathcal{Z}_2$ are routed to $t_Z = \mathrm{argmin}\{d(t) \mid t \in Z\}$ and x, respectively, the total amount of these flow on each edge of T_x is at most λ.* □

3.2 Algorithm for Tree Partition

In this subsection, we present an algorithm that exploits the results in Lemmas 3-5 to compute a partition of the source set of a general tree given in the next theorem.

Theorem 1. *Given a tree T rooted at s, an edge capacity $\lambda > 0$, a source set $S \subseteq V(T)$, a demand function $q : S \to R^+$ such that $q(t) \leq \lambda/2$, $t \in S$, and a vertex weight function $d : S \to R^+$, there is a partition $\mathcal{S} = \mathcal{S}_1 \cup \mathcal{S}_2 \cup \mathcal{S}_3 \cup \mathcal{S}_4$ of S, where $\mathcal{S}_3 = \cup_{1 \leq i \leq k}\{X_i, Y_i\}$ and $\mathcal{S}_4 = \cup_{1 \leq i \leq \ell}\{A_i, B_i, C_i\}$, and a set $\mathcal{H} = \{t_Z \in S \mid Z \in \mathcal{S}\}$ of hub vertices, that satisfy:*

(i) *For each subset $Z \in \mathcal{S}_1$, $q(Z) < (4/7)\lambda$ and $t_Z = s$.*
(ii) *For each subset $Z \in \mathcal{S}_2$, $q(Z) \geq (4/7)\lambda$ and $t_Z = \mathrm{argmin}\{d(t) \mid t \in Z\}$.*
(iii) *For $i = 1, 2, \ldots, k$, $q(Y_i) \geq (4/7)\lambda$, $q(X_i \cup Y_i) \geq (8/7)\lambda$, $t_{X_i} = \mathrm{argmin}\{d(t) \mid t \in X_i \cup Y_i\}$, and $t_{Y_i} = \mathrm{argmin}\{d(t) \mid t \in Y_i\}$.*
(iv) *For $i = 1, 2, \ldots, \ell$, $q(A_i \cap Z'_x), q(B_i \cap Z'_x) > (3/7)\lambda$, and $q(C_i \cap Z'_x) \geq (5/7)\lambda$, where $Z'_x \subseteq A_i \cup B_i \cup C_i$ with $q(Z'_x) \geq (12/7)\lambda$, and $t_{A_i} = \mathrm{argmin}\{d(t) \mid t \in Z'_x\}$, $t_{B_i} = \mathrm{argmin}\{d(t) \mid t \in Z'_x - A_i\}$, and $t_{C_i} = \mathrm{argmin}\{d(t) \mid t \in C_i \cap Z'_x\}$.*
(v) *When the total demand of each subset $Z \in \mathcal{S}$ is routed to t_Z simultaneously, the total amount of these flow on each edge of T is bounded from above by λ.*

Furthermore, such a partition $\mathcal{S}$ can be computed in polynomial time. □

To prove Theorem 1, we can assume without loss of generality that in a given tree T, (i) all sources are leaves, i.e., $S = L(T)$, by introducing a new edge of weight zero for each non-leaf source, and (ii) $|Ch(v)| = 2$ holds for every non-leaf $v \in V(T)$, i.e., T is a binary tree rooted at s, by splitting vertices of degree more than 3 with new edges of zero weights.

We prove Theorem 1 by showing that the next algorithm actually delivers a desired partition $\mathcal{S} = \mathcal{S}_1 \cup \mathcal{S}_2 \cup \mathcal{S}_3 \cup \mathcal{S}_4$. We first choose a vertex $v \notin Q \cup \{s\}$ with the maximum depth in the current tree such that the total demand of a source set Z_v of the tree rooted at v is at least $(4/7)\lambda$, where Q is initialized to be empty and is used to keep track of vertices v in the current tree such that T_v contains a balance-tree and satisfies $q(Z_v) < (8/7)\lambda$. Depending on the total demand of Z_v, we add Z_v to $\mathcal{S}_2$, add v to Q, or compute a partition of Z_v by using Lemma 3 or 4. In the latter case, we add the subsets of the obtained partition to one of $\mathcal{S}_3$ or $\mathcal{S}_4$. We then remove all sources in $\mathcal{S}_2 \cup \mathcal{S}_3 \cup \mathcal{S}_4$ from S and repeat these steps on the minimal subtree of T that spans s and the current source set until there is no such vertex v. Finally, we partition the remaining set of sources by using Lemma 5 and add the resulting partition to one of $\mathcal{S}_1$ or $\mathcal{S}_2$. A formal description of the algorithm is the following.

Algorithm. TreePartition
Input: A binary tree $\widehat{T}$ rooted at s, a capacity λ of each edge, a set $S = L(\widehat{T})$ of sources, a demand function $q : S \to R^+$ such that $q(t) \le \lambda/2$, $t \in S$, and a vertex weight function $d : S \to R^+$.
Output: A pair $(\mathcal{S}, \mathcal{H})$ that satisfies the conditions in Theorem 1.
Initialize $T := \widehat{T}$; $Q := \mathcal{H} := \mathcal{S}_1 := \mathcal{S}_2 := \mathcal{S}_3 := \mathcal{S}_4 := \emptyset$.
1 **while** there exists a vertex $v \in V(T) - \{s\} - Q$ such that
$q(V(T_v) \cap S) \ge (4/7)\lambda$ **do**
2 Choose such v with the maximum depth from s;
3 Let $Z_v := D_T(v) \cap S$; $T_v := T\langle Z_v \rangle$;
4 **begin** /* Distinguish the next four cases. */
5 **Case-1** $q(Z_v) \le \lambda$: Let $\mathcal{S}_2 := \mathcal{S}_2 \cup \{Z_v\}$;
6 $t_{Z_v} = \text{argmin}\{d(t) \mid t \in Z_v\}$; $\mathcal{H} := \mathcal{H} \cup \{t_{Z_v}\}$;
7 **Case-2** $\lambda < q(Z_v) < (8/7)\lambda$: Let $Q := Q \cup \{v\}$;
8 **Case-3** $(8/7)\lambda \le q(Z_v) < (12/7)\lambda$:
9 Apply Lemma 3 to $(T_v, Z_v, q, d, \lambda)$ to get a partition $\{X, Y\}$ of Z_v and vertices t_X and t_Y that satisfy the conditions in the lemma;
10 $\mathcal{S}_3 := \mathcal{S}_3 \cup \{X, Y\}$; $\mathcal{H} := \mathcal{H} \cup \{t_X, t_Y\}$
11 **Case-4** $(12/7)\lambda \le q(Z_v) < (16/7)\lambda$:
12 Apply Lemma 4 to $(T_v, Z_v, q, d, \lambda)$ to get a partition $\{A, B, C\}$ of Z_v and vertices t_A, t_B, and t_C that satisfy the conditions in the lemma;
13 $\mathcal{S}_4 := \mathcal{S}_4 \cup \{A, B, C\}$ and $\mathcal{H} := \mathcal{H} \cup \{t_A, t_B, t_C\}$
14 **end**; /* Cases-1,2,3,4 */
15 Let $S := S - (\mathcal{S}_2 \cup \mathcal{S}_3 \cup \mathcal{S}_4)$; $T := T\langle S \cup \{s\} \rangle$
16 **endwhile**;
17 **if** $S \ne \emptyset$
18 Regard T as a tree T_s rooted at s and apply Lemma 5 to (T_s, S, q, d, λ)

to get a partition $\mathcal{Z}_1 \cup \mathcal{Z}_2$ of S and a vertex t_Z for each $Z \in \mathcal{Z}_1 \cup \mathcal{Z}_2$ that satisfy the conditions in the lemma, where $t_Z = s$ for each $Z \in \mathcal{Z}_2$;
19 $\mathcal{S}_1 := \mathcal{Z}_2$; $\mathcal{S}_2 := \mathcal{S}_2 \cup \mathcal{Z}_1$; $\mathcal{H} := \mathcal{H} \cup \{t_Z \mid Z \in \mathcal{Z}_1 \cup \mathcal{Z}_2\}$
20 **endif**.

Proof of Theorem 1. We first prove by induction the correctness of algorithm TreePartition. We first consider the vertex v chosen in the first iteration of the while-loop. By the choice of v, $q(V(T_u) \cap S) < (4/7)\lambda$ for all $u \in Ch(v)$. Hence $(4/7)\lambda \leq q(Z_v) < (8/7)\lambda$ holds, which implies that $q(Z_v) \leq \lambda$ or $\lambda < q(Z_v) < (8/7)\lambda$ can occur in the first iteration. If $q(Z_v) \leq \lambda$ holds, then Z_v is removed from S and added to $\mathcal{S}_2$. Otherwise $\lambda < q(Z_v) < (8/7)\lambda$ holds and hence T_v is a balance-tree. In the latter case, v is added to a set Q.

Assume that the algorithm works correctly after the execution of the jth iteration, and let T be the current tree. We show the correctness of the algorithm during the execution of the $(j+1)$th iteration. Note that, for any vertex v chosen by the algorithm, Z_v will be removed from the current S except for the case where $\lambda < q(Z_v) < (8/7)\lambda$. Now let v be a vertex selected in the $(j+1)$st iteration. Then we see that, for each $u \in Ch(v)$, either (i) $q(V(T_u) \cap S) < (4/7)\lambda$ holds (if u has not been chosen before by the algorithm) or (ii) $u \in Q$ holds and T_u contains a balance-tree and satisfies $q(V(T_u) \cap S) < (8/7)\lambda$ (otherwise). Therefore, one of $(4/7)\lambda \leq q(Z_v) \leq \lambda$, $\lambda < q(Z_v) < (8/7)\lambda$, $(8/7)\lambda \leq q(Z_v) < (12/7)\lambda$, and $(12/7)\lambda \leq q(Z_v) < (16/7)\lambda$ holds. Let B_v^1 and B_v^2 denote the two branches of T_v, and let Z_v^i denote the set of sources in B_v^i, $i = 1, 2$, where $q(Z_v^1) \geq q(Z_v^2)$. Now if $q(Z_v) \leq \lambda$ holds, then Z_v is removed from the current S after it is added to $\mathcal{S}_2$. If $\lambda < q(Z_v) < (8/7)\lambda$ holds, then T_v is a balance-tree (if $q(Z_v^1), q(Z_v^2) < (4/7)\lambda$) or B_v^1 (consequently T_v) contains a balance-tree (by $q(Z_v^1) \geq q(Z_v^2)$). In this case, v is added to a set Q. Finally, if $(8/7)\lambda \leq q(Z_v) < (12/7)\lambda$ (resp., $(12/7)\lambda \leq q(Z_v) < (16/7)\lambda$) holds then T_v satisfies conditions of Lemma 3 (resp., Lemma 4) in this case. In the latter two cases, Z_v is removed from the current S after elements of its partition are added to appropriate subsets of $\mathcal{S}$. Therefore, the algorithm works correctly during the execution of all iterations of the while-loop.

After the final iteration, there is no vertex $v \in V(T) - \{s\} - Q$ such that $q(V(T_v) \cap S) \geq (4/7)\lambda$ for the current tree T. If the current $S \neq \emptyset$, then for each $u \in Ch(s)$, either (i) $q(V(T_u) \cap S) < (4/7)\lambda$ holds (if u has not been chosen before by the algorithm) or (ii) $u \in Q$ holds and T_u contains a balance-tree and satisfies $q(V(T_u) \cap S) < (8/7)\lambda$ (otherwise). That is, the current tree T satisfies the conditions in Lemma 5 and a desired partition of the current S can be constructed.

Now we prove that the partition obtained from algorithm TreePartition satisfies conditions (i)-(v) in Theorem 1. Conditions (i)-(iv) follow immediately from construction of $\mathcal{S}_1$, $\mathcal{S}_2$, $\mathcal{S}_3$, and $\mathcal{S}_4$. Now we show (v). Let v be the vertex chosen in line 2 of an arbitrary iteration of the algorithm, where the subtree T_v of the current tree T is being processed in this iteration. Now, if Case-2 holds, then we just add v to Q and then move to the next iteration (the current S and T remain unchanged in this iteration). Otherwise (Case-1, 3, or 4 holds),

the algorithm partitions the set Z_v of all sources of T_v into subsets and chooses a hub vertex from each of these subsets. We then remove Z_v from the current source set S, that is, none of the vertices of T_v will become a hub vertex in the subsequent iterations of the algorithm. Thus it is sufficient to show that, overall iterations of the algorithm, when the demand of each source in Z_v is routed to its hub vertex simultaneously, the total flow on each edge of T_v is bounded from above by λ. Hence (v) follows from the conditions of Lemmas 3, 4, and 5. This completes the correctness of TreePartition and the proof of Theorem 1. □

4 Approximation Algorithm to MCEI

This section describes a framework of our approximation algorithm for the MCEI and then analyzes its approximation ratio. The algorithm relies on the results on tree partition we provided in Section 3.

The basic idea of the algorithm is to first produce a tree T of minimum cost including all vertices in $S \cup \{s\}$. For each source $t \in S$ with $q(t) > \lambda/2$, we install a copy of each edge in a shortest path $SP(s,t)$ between s and t in (G,w), and let $P_t := SP(s,t)$. We then find a partition $\mathcal{S}$ of the remaining sources in S, and assign a hub vertex t_Z for each subset $Z \in \mathcal{S}$, such that when the total demand of each subset is routed to its hub vertex simultaneously, the amount of these flow on each edge of T is at most λ. Finally, for each set $Z \in \mathcal{S}$, we install a copy of each edge in a shortest path $SP(s,t_Z)$ between s and t_Z in (G,w), and construct a path P_t from the path between t and t_Z in T by adding $SP(s,t_Z)$ for all $t \in Z$.

Algorithm. ApproxMCEI
Input: An instance $I = (G = (V,E), w, S, q, s, \lambda)$ of the MCEI.
Output: A solution $\mathcal{P}$ to I.

Step 1. Compute a Steiner tree T that spans $S \cup \{s\}$ in G.
Regard T as a tree rooted at s, and define $d : S \to R^+$ by setting

$$d(t) := d_{(G,w)}(s,t), \quad t \in S.$$

Step 2. Let $S' = \{t \in S \mid q(t) > \lambda/2\}$.
For each $t \in S'$, choose a shortest path $SP(s,t)$ between s and t in (G,w), join t to s by installing a copy of each edge in $SP(s,t)$, and let $P_t := SP(s,t)$.
Step 3. Apply Theorem 1 to $(T, S-S', q, s, d, \lambda)$ to obtain a partition

$$\mathcal{S} = \mathcal{S}_1 \cup \mathcal{S}_2 \cup \mathcal{S}_3 \cup \mathcal{S}_4$$

of $S - S'$, where $\mathcal{S}_3 = \cup_{1 \le i \le k}\{X_i, Y_i\}$ and $\mathcal{S}_4 = \cup_{1 \le i \le \ell}\{A_i, B_i, C_i\}$, and a set $\mathcal{H} = \{t_Z \in S \mid Z \in \mathcal{S}\}$ of hub vertices, that satisfy conditions (i)-(v) of the theorem.
Step 4. For each $t \in Z \in \mathcal{S}_1$, let P_t be the path between t and s in T.
For each $Z \in \mathcal{S} - \mathcal{S}_1$,

Choose a shortest path $SP(s, t_Z)$ between s and t_Z in (G, w) and join t_Z to s by installing a copy of each edge in $SP(s, t_Z)$.
For each $t \in Z$, let P_t be the path obtained from the path between t and t_Z in T by adding $SP(s, t_Z)$.

Step 5. Output $\mathcal{P} = \{P_t \mid t \in S\}$. □

Before analyzing the worst case performance of this algorithm, we show the following lemma.

Lemma 6. *Let $\mathcal{S} = \mathcal{S}_1 \cup \mathcal{S}_2 \cup \mathcal{S}_3 \cup \mathcal{S}_4$ be a partition from a tree $\widehat{T}$ by algorithm* TreePartition *and let $\mathcal{H} = \{t_Z \in S \mid Z \in \mathcal{S}\}$ be the associated set of hub vertices. Then, we have*

$$\sum_{t \in Z \in \mathcal{S}_2 \cup \mathcal{S}_3 \cup \mathcal{S}_4} q(t)d(t) \geq (4/7)\lambda \sum_{Z \in \mathcal{S}_2 \cup \mathcal{S}_3 \cup \mathcal{S}_4} d(t_Z).$$

Proof. First, consider a subset $Z \in \mathcal{S}_2$. Condition (ii) of Theorem 1 implies that

$$\sum_{t \in Z} q(t)d(t) \geq q(Z)d(t_Z) \geq (4/7)\lambda d(t_Z). \tag{1}$$

Now consider a pair of subsets $X_i, Y_i \in \mathcal{S}_3$ defined in Theorem 1(iii). Let $q(Y_i) = q_1 + q_2$ such that $q_1 = (4/7)\lambda$ ($q(Y_i) \geq (4/7)\lambda$). Hence, we have

$$\begin{aligned}\sum_{t \in X_i} q(t)d(t) + \sum_{t \in Y_i} q(t)d(t) &\geq q(X_i)d(t_{X_i}) + (4/7)\lambda d(t_{Y_i}) + q_2 d(t_{Y_i}) \\ &= (q(X_i) + q_2)d(t_{X_i}) + (4/7)\lambda d(t_{Y_i}) \\ &\geq (4/7)\lambda(d(t_{X_i}) + d(t_{Y_i})),\end{aligned} \tag{2}$$

since $q(X_i \cup Y_i) \geq (8/7)\lambda$ and $d(t_{X_i}) \leq d(t_{Y_i})$.

Finally, consider a triple of subsets $A_i, B_i, C_i \in \mathcal{S}_4$ defined in Theorem 1(iv). Let $q(C_i \cap Z'_x) = q_1 + q_2 + q_3$ such that $q_1 = (4/7)\lambda$ and $q(A_i \cap Z'_x) + q_2$,$q(B_i \cap Z'_x) + q_3 \geq (4/7)\lambda$. Note that, Condition (iv) of Theorem 1 implies that q_1, q_2, and q_3 are well defined. Hence, we have

$$\begin{aligned}&\sum_{t \in A_i} q(t)d(t)) + \sum_{t \in B_i} q(t)d(t) + \sum_{t \in C_i} q(t)d(t) \\ &\geq \sum_{t \in A_i \cap Z'_x} q(t)d(t) + \sum_{t \in B_i \cap Z'_x} q(t)d(t) + \sum_{t \in C_i \cap Z'_x} q(t)d(t) \\ &\geq q(A_i \cap Z'_x)d(t_{A_i}) + q(B_i \cap Z'_x)d(t_{B_i}) + (4/7)\lambda d(t_{C_i}) + (q_2 + q_3)d(t_{C_i}) \\ &\geq (4/7)\lambda(d(t_{A_i}) + d(t_{B_i}) + d(t_{C_i})),\end{aligned} \tag{3}$$

since $d(t_{A_i}) \leq d(t_{B_i}) \leq d(t_{C_i})$.

Hence the proof completes by summing inequalities (1), (2), and (3) overall subsets in $\mathcal{S}_2 \cup \mathcal{S}_3 \cup \mathcal{S}_4$. □

We now turn to proving that the solution output from algorithm ApproxMCEI is within a factor of $(15/8 + \rho_{\mathrm{ST}})$ of the optimal solution.

Theorem 2. *For an instance $I = (G = (V, E), w, S, q, s, \lambda)$ of the* MCEI*, algorithm* ApproxMCEI *delivers a $(15/8 + \rho_{\mathrm{ST}})$-approximate solution $\mathcal{P}$, where ρ_{ST} is the performance ratio for approximating Steiner tree problem.*

Proof. Let $opt(I)$ denote the weight of an optimal solution. By the construction, the cost of $\mathcal{P}$ is bounded by

$$cost(\mathcal{P}) \leq w(T) + \sum_{t \in S'} d(t) + \sum_{Z \in \mathcal{S}_2 \cup \mathcal{S}_3 \cup \mathcal{S}_4} d(t_Z).$$

For a minimum Steiner tree T^* that spans $S \cup \{s\}$, we have $w(T) \leq \rho_{\mathrm{ST}} w(T^*)$ and $w(T^*) \leq opt(I)$ by Lemma 1. Hence $w(T) \leq \rho_{\mathrm{ST}} \cdot opt(I)$ holds. To prove the theorem, it suffices to show that

$$\sum_{t \in S'} d(t) + \sum_{Z \in \mathcal{S}_2 \cup \mathcal{S}_3 \cup \mathcal{S}_4} d(t_Z) \leq (15/8) opt(I). \tag{4}$$

To prove this inequality, we distinguish two different cases. In the first case, $\sum_{t \in S'} q(t)d(t) \geq \sum_{t \in S - S'} q(t)d(t)$. By Lemma 1, this implies that

$$\begin{aligned} opt(I) \geq (1/\lambda) \sum_{t \in S} q(t)d(t) &= (1/\lambda)\Big(\sum_{t \in S'} q(t)d(t) + \sum_{t \in S - S'} q(t)d(t)\Big) \\ &\geq (2/\lambda) \sum_{t \in S - S'} q(t)d(t) \\ &\geq (2/\lambda) \sum_{t \in Z \in \mathcal{S}_2 \cup \mathcal{S}_3 \cup \mathcal{S}_4} q(t)d(t) \\ &\geq (8/7) \sum_{Z \in \mathcal{S}_2 \cup \mathcal{S}_3 \cup \mathcal{S}_4} d(t_Z), \end{aligned} \tag{5}$$

where the last inequality follows from Lemma 6. Inequality (5) and Lemma 2 prove (4) in this case.

In the second case, $\sum_{t \in S'} q(t)d(t) < \sum_{t \in S - S'} q(t)d(t)$. Then it is easy to see that there exist two real numbers $0 \leq \alpha, \beta \leq 1$ such that $\alpha + \beta = 1$, $\alpha < \beta$, $(1/\lambda) \sum_{t \in S'} q(t)d(t) \leq \alpha opt(I)$, and $(1/\lambda) \sum_{t \in S - S'} q(t)d(t) \leq \beta opt(I)$. Since $q(t) > \lambda/2$ for all $t \in S'$, we have

$$(1/2) \sum_{t \in S'} d(t) < (1/\lambda) \sum_{t \in S'} q(t)d(t) \leq \alpha opt(I). \tag{6}$$

On the other hand, Lemma 6 implies that

$$(4/7) \sum_{Z \in \mathcal{S}_2 \cup \mathcal{S}_3 \cup \mathcal{S}_4} d(t_Z) \leq (1/\lambda) \sum_{t \in Z \in \mathcal{S}_2 \cup \mathcal{S}_3 \cup \mathcal{S}_4} q(t)d(t) \leq \beta opt(I). \tag{7}$$

By multiplying (6) and (7) by 2 and 7/4, respectively, and adding the obtained inequalities, we have

$$\sum_{t \in S'} d(t) + \sum_{Z \in \mathcal{S}_2 \cup \mathcal{S}_3 \cup \mathcal{S}_4} d(t_Z) \leq (2\alpha + (7/4)\beta) opt(I) < (15/8) opt(I),$$

by the assumptions on α and β. □

5 Concluding Remarks

In this paper, we have studied the minimum cost edge installation problem (MCEI), a problem of finding a routing from a set of sources to a single sink in networks. We have designed a $(15/8 + \rho_{\text{ST}})$-approximation algorithm for the MCEI based on an elaborate tree partition, where ρ_{ST} is any approximation ratio achievable for the Steiner tree problem.

References

1. Awerbuch, B., Azar, Y.: Buy-at-bulk network design. In: EEE Symposium on Foundations of Computer Science (FOCS), pp. 542–547 (1997)
2. Grandoni, F., Italiano, G.F.: Improved approximation for single-sink buy-at-bulk. In: Asano, T. (ed.) ISAAC 2006. LNCS, vol. 4288, pp. 111–120. Springer, Heidelberg (2006)
3. Guha, S., Meyerson, A., Munagala, K.: A constant factor approximation for the single sink edge installation problem. In: ACM symposium on the Theory of Computing (STOC), pp. 383–388 (2001)
4. Hassin, R., Ravi, R., Salman, F.S.: Approximation algorithms for a capacitated network design problem. Algorithmica 38, 417–431 (2004)
5. Khuller, S., Raghavachari, B., Young, N.N.: Balancing minimum spanning and shortest path trees. Algorithmica 14, 305–322 (1993)
6. Mansour, Y., Peleg, D.: An approximation algorithm for minimum-cost network design. Tech. Report Cs94-22, The Weizman Institute of Science, Rehovot (1994); also presented at the DIMACS Workshop on Robust Communication Network (1998)
7. Robins, G., Zelikovsky, A.Z.: Improved Steiner tree approximation in graphs. In: Proceedings of the 11th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2000), pp. 770–779 (2000)
8. Salman, F.S., Cheriyan, J., Ravi, R., Subramanian, S.: Approximating the single-sink link-installation problem in network design. SIAM J. Optim. 11, 595–610 (2000)
9. Zheng, H.-Z., Chu, D.-H., Zhan, D.-C.: Effective algorithm CFL based on Steiner tree and UFL problem. IJCSNS 6(9A), 24–27 (2006)

Approximability of Packing Disjoint Cycles

Zachary Friggstad* and Mohammad R. Salavatipour**

Department of Computing Science,
University of Alberta,
Edmonton, Alberta T6G 2E8, Canada
{zacharyf,mreza}@cs.ualberta.ca

Abstract. Given a graph G, the edge-disjoint cycle packing problem is to find the largest set of cycles of which no two share an edge. For undirected graphs, the best known approximation algorithm has ratio $O(\sqrt{\log n})$ [14,15]. In fact, they proved the same upper bound for the integrality gap of this problem by presenting a simple greedy algorithm. Here we show that this is almost best possible. By modifying integrality gap and hardness results for the edge-disjoint paths problem [1,9], we show that the undirected edge-disjoint cycle packing problem has an integrality gap of $\Omega(\frac{\sqrt{\log n}}{\log\log n})$ and furthermore it is quasi-NP-hard to approximate the edge-disjoint cycle problem within ratio of $O(\log^{\frac{1}{2}-\epsilon} n)$ for any constant $\epsilon > 0$. The same results hold for the problem of packing vertex-disjoint cycles.

1 Introduction

In the problem of *edge-disjoint cycle packing* (EDC) we are given a graph G and our goal is to find a largest set of edge-disjoint cycles. The vertex analog of the problem, *vertex-disjoint cycle packing* (VDC), is the problem of finding a largest set of vertex-disjoint cycles in the given graph. The EDC problem has been studied extensively in both directed and undirected settings (e.g. see Balister et al. [3], Caprara et al.[5], and Seymour [18]). A discussion on the applications of packing cycles to computational biology and reconstructing evolutionary trees can be found in [3]. Both EDC and VDC are known to be NP-hard even for undirected graphs and for very restricted cases of the problem (see e.g. [10]). This motivates the study of approximation algorithms for these problems. Caprara, Panconesi and Rizzi [5] showed that EDC is APX-hard even when restricted on planar graphs. They also presented a simple greedy algorithm with approximation ratio $O(\log n)$. Recently, Krivelevich et al. [14,15] showed that a modification of the simple greedy algorithm of [5] with a more careful analysis yields an $O(\sqrt{\log n})$-approximation for EDC on undirected graphs. In fact, the algorithm obtains an integer solution that is within factor $O(\sqrt{\log n})$ of the optimal fractional solution. They showed examples for which the solution obtained

* Supported by NSERC.
** Supported by NSERC and a University Start-up fund.

T. Tokuyama (Ed.): ISAAC 2007, LNCS 4835, pp. 304–315, 2007.

by the greedy algorithm was within $\Omega(\sqrt{\log n})$ of the optimal solution but it falls short of proving any super-constant lower bound on the integrality gap or approximability of the problem. They also presented an $O(\sqrt{n})$-approximation for EDC on directed graphs and an $O(\log n)$-approximation for undirected VDC. Subsequently, in [16,15], an integrality gap of $\Omega(\frac{\log n}{\log \log n})$ for EDC on directed graphs was proved. This result was followed by a hardness of approximation. There was proved that unless NP $\subseteq$ DTIME($n^{\text{polylog}(n)}$), it is hard to approximate EDC on directed graphs within $O(\log^{1-\epsilon} n)$ for any $\epsilon > 0$. However, the best known lower bound on the approximability of EDC on undirected graphs remains APX-hardness and the best lower bound for integrality gap is $O(1)$. For EDC on planar graphs, Caprara, Panconesi, and Rizzi give a $2+\epsilon$-approximation algorithm [4].

For the related problem of edge-disjoint paths (EDP), on directed graphs the best approximation algorithms have ratio $O(\min\{n^{\frac{2}{3}} \log^{\frac{1}{3}} n, \sqrt{m}\})$ [6,13,19] and it is known the problem is hard to approximate within $O(m^{\frac{1}{2}-\epsilon})$ for any $\epsilon > 0$ [11]. For undirected graphs, the best known approximation ratio for EDP is $O(\sqrt{n})$ [7] whereas the best known hardness result is only $\Omega(\log^{\frac{1}{2}-\epsilon} n)$ for any $\epsilon > 0$ [1,9]. The latter result was built on the major advance on the lower bound of EDP (from APX-hardness to $\Omega(\log^{\frac{1}{3}-\epsilon} n)$) by Andrews and Zhang [2].

In this paper, we improve the lower bounds for both EDC and VDC. More specifically, we first present an integrality gap construction which shows that the integrality gap upper bound of [14,15] is almost tight.

Theorem 1. *The integrality gap of EDC on undirected graphs is* $\Omega(\frac{\sqrt{\log n}}{\log \log n})$.

Then we show almost the same bound for the hardness of approximation.

Theorem 2. *The EDC problem on undirected graphs is hard to approximate within* $O(\log^{\frac{1}{2}-\epsilon} n)$ *for any* $\epsilon > 0$ *unless* NP $\subseteq$ ZPTIME($n^{\text{polylog}(n)}$).

This shows that the simple greedy algorithm of [14,15] with approximation ratio $O(\sqrt{\log n})$ is almost best possible for EDC. The reduction in the proof of Theorem 2 works, without modification, to prove the same hardness result for VDC. Our results are heavily motivated by the hardness of the edge-disjoint paths problem presented by Chuzhoy and Khanna in [9]. Nevertheless, they show a rather surprising approximability threshold for a very natural packing problem. In fact there are very few problems known to have a sub-logarithmic approximability threshold ([8,12]). One other important message to be taken from our results is that, in order to improve the hardness of approximation for EDP (from $\Omega(\log^{\frac{1}{2}-\epsilon} n)$ to beyond $\Omega(\sqrt{\log n})$), there has to be substantially new ideas developed that exploit the differences between EDC and EDP problems; since such a hardness result should not be adaptable to work for EDC (because we already have an $O(\sqrt{\log n})$-approximation for EDC).

The rest of the paper is separated into three more sections. In Section 2 we describe the construction of a graph with large integrality gap for EDC. The ideas from this section motivate the proof of hardness of approximation result in the subsequent section.

2 Integrality Gap

The construction of an instance of the EDC problem with a large integrality gap is similar to the construction used by Chuzhoy and Khanna in [9]. We begin by generating a random graph G and use this graph to generate another graph H. With sufficiently large probability, the resulting graph H has a super-constant integrality gap. We use a model of random graph different from [9] which enables us to handle some tricky special cases that are overlooked in the analysis of [9].

2.1 Constructing the Gap

Given a sufficiently large integer n, define $\beta_1 = \frac{\sqrt{\log n}}{8 \log \log n}$ and $\beta_2 = 5\beta_1 \ln \beta_1$. The ultimate goal is to construct a graph with $O(n^2)$ nodes and integrality gap $\Omega(\beta_1)$. Start by building a random Hamiltonian cycle F on n vertices. Then we add a random graph $G_1 = G_{n,p}$ to F with $p = \frac{2\beta_2}{n-1}$, i.e. for each pair of nodes, if there is no edge between them already (due to F), we add it randomly (and independently) with probability $p = \frac{2\beta_2}{n-1}$. This is our graph G.

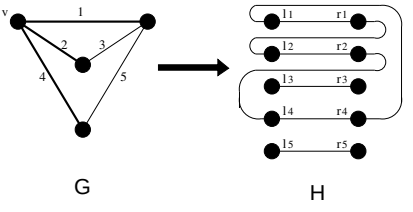


Fig. 1. Constructing the canonical cycle for v

Now, from graph G we will create another graph H as follows. For each edge $e_i \in G$, add vertices ℓ_i and r_i to H and connect them with an edge $\ell_i r_i$ which will be called a *special* edge. Finally, for each vertex v of G, let $e_{v_1}, e_{v_2}, \ldots, e_{v_k}$ be the edges incident with v in some arbitrary order. Add edges $r_{v_i}\ell_{v_{i+1}}$ to H for all $1 \leq i \leq k$ where $v_{k+1} = v_1$. Call the sequence of vertices $\ell_{v_1}, r_{v_1}, \ell_{v_2}, r_{v_2}, \ldots, \ell_{v_k}, r_{v_k}$ the *canonical cycle* of v denoted by C_v. Notice that each special edge $\ell_i r_i$ in H (corresponding to edge $e_i \in G$) appears in exactly two canonical cycles C_u and C_v where u and v are the endpoints of e_i in the original graph G. Every other edge appears in exactly one canonical cycle in H. Note that since the minimum degree of G is 2, every vertex in G has a corresponding canonical cycle in H. So we have n canonical cycles in H.

2.2 Analysis

If we assign a fractional value of $\frac{1}{2}$ to each canonical cycle in H, each special edge has total fractional value 1 and all the other edges have fractional value $\frac{1}{2}$. Thus, no edge constraint is violated and we have a fractional packing of cycles with total value $n/2$ in H (as there are n canonical cycles in H).

We now bound the number of cycles in any integral packing $\mathcal{C}$ in H. First observe that the expected degree of each node v in G_1, $\mathrm{E}[d(v)]$, is $2\beta_2$. Using Chernoff bound, $\Pr[d(v) < \beta_2] \leq \Pr[d(v) < \frac{1}{2}\mathrm{E}[d(v)]] \leq e^{-\beta_2/8}$. So the expected number of nodes with degree smaller than β_2 in G_1 is at most $n \cdot e^{-\beta_2/8} \leq \frac{n}{8\beta_1}$. Thus using Markov's inequality, with probability at least $\frac{7}{8}$ the number of nodes with degree smaller than β_2 in G_1, and therefore in G, is at most $\frac{n}{\beta_1}$.

Second, let $M_1 = |E(G_1)|$. Since $\mathrm{E}[M_1] = p \cdot \binom{n}{2} = \beta_2 n$, again using Chernoff bound, the probability of $|M_1 - \beta_2 n| > \beta_2 n/4$ is exponentially small. So we can assume that with probability at least $\frac{7}{8}$:

$$\frac{3}{4}\beta_2 n < M_1 < \frac{5}{4}\beta_2 n. \tag{1}$$

If we define $M = |E(G)|$, since $M_1 \leq M \leq M_1 + n$, using (1), the probability of event $|M - \beta_2 n| > \beta_2 n/2$ is at most $\frac{1}{8}$. Let the bad event $\mathcal{E}_0$ be the event that either $|M - \beta_2 n| > \beta_2 n/2$ or there are more than $\frac{n}{\beta_1}$ nodes with degree smaller than β_2. From above it follows with probability at least $\frac{3}{4}$ event $\mathcal{E}_0$ does not happen. Note that for every pair of nodes uv, for the probability of having an edge $e = uv$ in G we have:

$$\Pr[e \notin G] = \Pr[e \notin F] \cdot \Pr[e \notin G_1 | e \notin F] = \left(1 - \frac{2(n-2)!}{(n-1)!}\right) \cdot \left(1 - \frac{2\beta_2}{n-1}\right)$$
$$= \left(1 - \frac{2}{n-1}\right) \cdot \left(1 - \frac{2\beta_2}{n-1}\right) = 1 + \frac{4\beta_2}{(n-1)^2} - \frac{2\beta_2 + 2}{n-1}$$

Thus $\Pr[e \in G] = \frac{2\beta_2+2}{n-1} - \frac{4\beta_2}{(n-1)^2} \leq \frac{3\beta_2}{n-1}$. Defining $p' = \frac{3\beta_2}{n-1}$ we can assume each edge exists in G with probability at most p'. For $g = 6\beta_1\beta_2$, we say that a cycle is short if it is of length less than g and long otherwise. Let $\mathcal{C}_1$, $\mathcal{C}_2$, and $\mathcal{C}_3$ be the set of canonical cycles, long cycles, and short cycles of $\mathcal{C}$, respectively. So $\mathcal{C} = \mathcal{C}_1 \cup \mathcal{C}_2 \cup \mathcal{C}_3$. We will bound the size of each $\mathcal{C}_i$ by $O(n/\beta_1)$ which implies $|\mathcal{C}| \in O(n/\beta_1)$. Let the bad event $\mathcal{E}_1$ be the event that there are more than n/β_1 edge-disjoint canonical cycles in $\mathcal{C}$, i.e. $|\mathcal{C}_1| > n/\beta_1$.

Lemma 1. *The probability of bad event $\mathcal{E}_1$ happening is at most $\frac{1}{4}$.*

Proof. Recall that each canonical cycle in H corresponds to a vertex in G and two canonical cycles are edge-disjoint if and only if the corresponding vertices in G are non-adjacent. So it is enough to show that w.h.p. every set of vertices of size n/β_1 has some edges. First we obtain a bound on the probability that some fixed subset $S \subseteq V(G)$ of size n/β_1 doesn't contain any edge. Since each potential edge in G exists with probability at least $\frac{2\beta_2}{n-1}$, the probability that a fixed set S of size n/β_1 is empty is at most:

$$\left(1 - \frac{2\beta_2}{n-1}\right)^{\binom{n/\beta_1}{2}} \leq \left(1 - \frac{2\beta_2}{n}\right)^{n^2/(4\beta_1^2)} \leq e^{-\frac{\beta_2 n}{2\beta_1^2}}.$$

The number of such sets S is $\binom{n}{n/\beta_1} \leq (e\beta_1)^{n/\beta_1} \leq \beta_1^{2n/\beta_1}$; so by union bound the probability of having any set S of size n/β_1 that does not contain an edge is at most:

$$\beta_1^{\frac{2n}{\beta_1}} \cdot e^{\frac{-\beta_2 n}{4\beta_1^2}} \leq e^{\frac{n}{\beta_1}\left(2\ln\beta_1 - \frac{\beta_2}{2\beta_1}\right)} = e^{-\frac{n\ln\beta_1}{2\beta_1}} \leq \frac{1}{4}.$$

□

To bound $|\mathcal{C}_2|$, first observe that graph H has $3M$ edges. Since all cycles in $\mathcal{C}_2$ are of length at least g then $|\mathcal{C}_2|$ is easily bound by $\frac{3M}{g} \leq \frac{n}{\beta_1}$, assuming that $\mathcal{E}_0$ does not happen.

Now we bound the size of $\mathcal{C}_3$. First, obtain the multi-graph H' from H by contracting all special edges $\ell_i r_i$ to a single vertex u_{e_i}. So each such vertex u_{e_i} now corresponds to an edge e_i in G. If there are two edges between two nodes of H' then the only explanation can be that the edges come from the same canonical cycle corresponding to a degree 2 vertex of G. If we assume bad event $\mathcal{E}_0$ does not happen, there are at most $\frac{n}{\beta_1}$ cycles of length 2 in H'. Now we bound the number of cycles of length $3 \leq k < g$ in H'. It is easy to see that a bound on the number of cycles of length less than g in H' is an upper bound on the number of cycles of length less than g in G. Denote by $\mathcal{E}_2$ the event that there are more than $\frac{n}{\beta_1}$ simple cycles in H'.

Lemma 2. *The probability of bad event $\mathcal{E}_2$ occurring is at most $\frac{1}{4}$.*

Proof. We begin by bounding the expected number of cycles of some fixed length $3 \leq k < g$ in H'. Let $C = e_{i_1}, e_{i_2}, \ldots, e_{i_k}$ be an ordered sequence of edges forming a cycle in H' where $e_{i_j} = u_{i_j}u_{i_{j+1}}$ and all u_{i_j}'s are distinct for $1 \leq j \leq k$ where $i_{k+1} = i_1$. Denote the edge in G corresponding to u_{i_j} by h_{i_j} for all $1 \leq j \leq k$. By the construction of H', for each two consecutive nodes $u_{i_j}, u_{i_{j+1}}$ in C $(1 \leq j \leq k)$, the corresponding edges h_{i_j} and $h_{i_{j+1}}$ in G must be incident with a vertex i.e. have a common end-point (note that $h_{i_1}, \ldots, h_{i_k}$ do not necessarily form a cycle in G since, for example, h_{i_j}, $h_{i_{j+1}}$, and $h_{i_{j+2}}$ can all be incident to the same vertex). Given a sequence of pairs of nodes in G like $h_{i_1}, \ldots, h_{i_k}$, whose corresponding nodes in H' form a simple cycle like C, the probability that all pairs $h_{i_1}, \ldots, h_{i_k}$ are actually edges in G is at most $\left(\frac{3\beta_2}{n-1}\right)^k$. A loose upper bound on the number of such sequences of pairs of nodes in G (that correspond to a simple cycle in H') is $(2n)^k$ since once we select a pair of nodes, there are at most $2n-4$ other pairs of nodes, each of which has an end-point in common with the previous one. Also, every sequence of k edges in G, like $h_{i_1}, \ldots, h_{i_k}$ corresponds to a sequence of k nodes in H', say $u_{i_1}, \ldots, u_{i_k}$, and if this sequence forms a (simple) cycle then it forms at most 2^k cycles in H' because between every pair of nodes in H' there are at most two edges. Thus, the expected number of cycles of length k in H' is at most $2^k \cdot (2n)^k \cdot \left(\frac{3\beta_2}{n-1}\right)^k \leq (16\beta_2)^k$. Summing over all cycle lengths $3 \leq k < g$ gives an upper bound of $(16\beta_2)^g$ on the expected number of short (simple) cycles. By Markov's inequality, the probability that there are more than $4(16\beta_2)^g$ cycles of length $3 \leq k < g$ in H' is at most $\frac{1}{4}$. We show that this quantity is bound by n/β_1.

$$4(16\beta_2)^g \leq \beta_2^{2g} = e^{2g\ln\beta_2} = e^{12\beta_1\beta_2\ln\beta_2} \leq e^{60\beta_1^2\ln\beta_1\ln\beta_2} \leq e^{120\beta_1^2\ln^2\beta_1}$$
$$\leq e^{\frac{120\ln n}{64(\ln\ln n)^2}\cdot\frac{(\ln\ln n)^2}{4}} \leq e^{\frac{\ln n}{2}} \leq \frac{n}{\beta_1}.$$

Therefore, the probability that there are more than n/β_1 short (simple) cycles in $\mathcal{C}_3$ is at most $\frac{1}{4}$. □

Therefore, assuming that $\mathcal{E}_0$ and $\mathcal{E}_2$ do not happen $|\mathcal{C}_3| \leq 2n/\beta_1$. By union bound, the probability of $\mathcal{E}_0 \cup \mathcal{E}_1 \cup \mathcal{E}_2$ is at most $\frac{3}{4}$. So with probability at least $\frac{1}{4}$ graph G (and accordingly graph H) with the above properties exist and so any collection of disjoint cycles of H is of size at most $|\mathcal{C}| \leq |\mathcal{C}_1| + |\mathcal{C}_2| + |\mathcal{C}_3| \leq \frac{4n}{\beta_1}$. Since we can pack $n/2$ cycles fractionally in H, then the integrality gap is $\Omega(\beta_1) = \Omega\left(\frac{\sqrt{\log n}}{\log\log n}\right)$. The number of vertices of graph H is $N = 2M \in O(n^2)$. Therefore, the integrality gap in H is $\Omega\left(\frac{\sqrt{\log N}}{\log\log N}\right)$.

3 The Hardness Construction

In this section we prove Theorem 2. We show how a modification of the construction used to prove the hardness of approximating edge-disjoint paths by Chuzhoy and Khanna in [9] can be used to show the same hardness for EDC. Our starting point is a PCP characterization of NP introduced in [17].

3.1 A PCP Characterization of NP

To begin, we use the same PCP characterization of NP used in [9] which is a slight modification of the characterization obtained by Samorodnitsky and Trevisan in [17]. Let Φ be an instance of 3SAT with n variables. For any constant $k > 0$, consider a PCP verifier that uses $r = O(\log n)$ random bits and queries $q = k^2$ bits of a proof Π. Let R be a random string of length r and denote the indices of the bits of the proof that are read given the random string R as $b_1(R), \ldots, b_q(R)$. Define a *configuration* to be the tuple $(R, a_1, \ldots, a_q)$ where R is a random string of length r and $a_i = \Pi_{b_i(R)} \in \{0, 1\}$, for $1 \leq i \leq q$, are the values of the bits read in the proof. A configuration $(R, a_1, \ldots, a_q)$ is called *accepting* if the PCP verifier accepts upon using random string R and reading proof bits $a_1, \ldots, a_q$. It follows ([9]) from the construction of [17] that for every constant $k > 0$ and for sufficiently large constant $\beta >> k^2$ there exists a PCP verifier for Φ with the following properties:

- $\lambda r = O(\log n \log\log n)$ random bits are used with $r = O(\log n)$ and $\lambda = \frac{2\beta\log\log n}{k^2}$.
- Exactly $q = \lambda k^2 = O(\log\log n)$ bits of the proof are queried for each random string.

- If Φ is satisfiable, then there exists a proof Π such that the acceptance probability of the PCP verifier upon reading Π is at least $2^{-\lambda}$.
- If Φ is not satisfiable, then the acceptance probability of the PCP verifier upon reading Π is at most $2^{-\lambda k^2}$ for all proofs Π.
- Every random string R participates in $2^{\lambda(2k-1)}$ accepting configurations.
- For every random string R and for every $j = 1 \dots q$, the number of accepting configurations with $\Pi_{b_j(R)} = 0$ and the number of accepting configurations with $\Pi_{b_j(R)} = 1$ are equal.
- Let Z_j to be the set of all accepting configurations with $\Pi_j = 0$ and let O_j be the set of all accepting configurations with $\Pi_j = 1$. Let $n_j = |Z_j| = |O_j|$. Then $n_j \geq 2^{\lambda r/2}$.
- Let $\mathcal{A}$ be the set of all accepting configurations. Then $|\mathcal{A}| \leq 2^{\lambda r} \cdot 2^{2\lambda k}$.

For a given instance of Φ of 3SAT with n variables, we assume that V is a PCP verifier with aforementioned properties and we choose k to be a large enough constant.

3.2 The Bit Gadget

The basic construction here is identical to that of [9]. Let M and X be two parameters which will be specified later. We only note that X will be exponentially larger than M, i.e. $X >> 2^M$. For each proof bit Π_i, we construct a *bit gadget* $G(i)$ in the following manner. Recall that Z_i and O_i are the set of accepting configurations in which bit Π_i is zero and one, respectively. For each accepting configuration $\alpha \in Z_i \cup O_i$ and for each $1 \leq m \leq M+1$, we create X vertices $v_{x,m}(\alpha, i)$, for $1 \leq x \leq X$, called *level* m vertices. Let $Z_m(i) = \{v_{1,m}(\alpha, i), \dots, v_{X,m}(\alpha, i)\}$ be the set of level m vertices when $\alpha \in Z_i$. Similarly define $O_m(i)$ to be the set of level m vertices when $\alpha \in O_i$. Between levels m and $m+1$, for $1 \leq m \leq M$, create Xn_i vertices $L_m(i) = \{\ell_{1,m}(i), \dots, \ell_{Xn_i,m}(i)\}$ as well as Xn_i vertices $R_m(i) = \{r_{1,m}(i), \dots, r_{Xn_i,m}(i)\}$ where $n_i = |Z_i| = |O_i|$.

The edges in the bit-gadget are specified as follows. For each $1 \leq m \leq M$, create a random matching between the Xn_i level m vertices associated with some $\alpha \in Z_i$ and the vertices in $L_m(i)$. Similarly, create a random matching between the vertices in $R_m(i)$ and the Xn_i level $m+1$ vertices associated with some $\alpha \in Z_i$. Repeat the same process between vertices associated with some $\alpha \in O_i$. Finally, for each $1 \leq m \leq M$ and for each $1 \leq j \leq Xn_i$, join $\ell_{m,j}(i)$ and $r_{m,j}(i)$ with an edge which we call a *special* edge. Figure 2 illustrates this construction.

For each configuration $\alpha \in Z_i \cup O_i$, we define a canonical path $P_x(\alpha, i)$ for $1 \leq x \leq X$, as being the path

$$(v_{x_1,1}(\alpha, i), \ell_{a_1,1}(i), r_{a_1,1}(i), v_{x_2,2}(\alpha, i), \dots, \ell_{a_M,M}(i), r_{a_M,M}(i), v_{x_{M+1},M+1}(\alpha, i))$$

where the indices $x_1 = x$ and the remaining x_m, a_m indices are defined by the random matchings. Essentially, a canonical path corresponding to configuration α begins at one of the X vertices $v_{x,1}(\alpha, i)$ and follows the random matchings

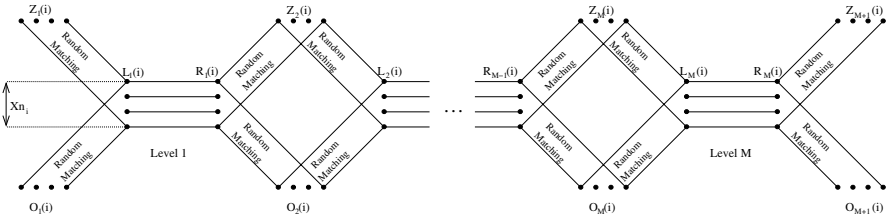


Fig. 2. Bit gadget construction for proof Π_i

between levels while never visiting vertices in O_i if $\alpha \in Z_i$ or never visiting vertices in Z_i if $\alpha \in O_i$.

Note that the canonical paths corresponding to the Xn_i configurations in Z_i are all edge-disjoint. Similarly, the canonical paths to the configurations in O_i are edge-disjoint. Each special edge belongs to exactly two canonical paths (one corresponding to a configuration in Z_i and one in O_i) and every other edge belongs to exactly one canonical path. Consider the set of Xn_i special edges at level m $(1 \leq m \leq M)$. Since each such special edge participates in exactly one canonical path representing a configuration in Z_i and one canonical path representing a configuration in O_i, the set of special edges in level m, defines a matching between canonical paths corresponding to configurations in Z_i and canonical paths corresponding to configuratinos in O_i. This matching is random (because of the random matchings placed before these special edges). So overall, the M levels of special edges define M random matchings between the canonical paths corresponding to configurations in Z_i and in O_i.

Let $\Delta = \frac{M}{8 \log M}$, noting that $M \geq 8\Delta \log \Delta$ holds. For each index i of proof Π, let $\mathcal{P}_0(i)$ be the set of canonical paths corresponding to a configuration in Z_i and $\mathcal{P}_1(i)$ be the set of canonical paths corresponding to a configuration in O_i. A bit gadget $G(i)$ is said to be *bad* if there is a pair of subsets $A \subseteq P_0(i)$, $B \subseteq P_1(i)$ with $|A| = |B| = \frac{Xn_i}{\Delta}$ such that all paths in $A \cup B$ are edge disjoint. Define bad event $\mathcal{B}_1$ to be the event that there is some bit gadget that is bad. The next lemma claims that with sufficiently high probability $\mathcal{B}_1$ does not happen. The proof is a simple first-moment analysis. The idea is that each path from A and each path from B can be matched by any of the M random matchings defined by the special edges, in which case the two paths are not edge-disjoint. Since our bit gadget is identical to the bit gadget constructed in [9], the following result holds as well.

Lemma 3. *[9] The probability that bad event $\mathcal{B}_1$ happens is at most $\frac{1}{\text{poly}(n)}$.*

3.3 The Main Construction

Here we show how to combine the bit gadgets into the final construction. This is essentially the same construction as in [9] with the modification that the corresponding source-sink pairs are connected by a new set of edges, called back-edges.

Let $\alpha = (R, a_{i_1}, a_{i_2}, \ldots, a_{i_q})$ be an accepting configuration with $i_1, \ldots, i_q$ being the indices of the proof bits queried upon reading the random string R. For each $1 \leq j < q$, we connect bit gadget $G(i_j)$ to bit gadget $G(i_{j+1})$ by creating a

random matching between the sets of vertices $\{v_{1,M+1}(\alpha, i_j) \dots v_{X,M+1}(\alpha, i_j)\}$ and $\{v_{1,1}(\alpha, i_{j+1}), \dots v_{X,1}(\alpha, i_{j+1})$. For each $1 \leq x \leq X$, we define canonical path $P_x(\alpha) = (P_{x_1}(\alpha, i_1), \dots, P_{x_q}(\alpha, i_q))$ where the x_j's are recursively defined as follows: $x_1 = x$ and x_j corresponds to the canonical path in $G(i_j)$ whose start point is matched with the end-point of $P_{x_{j-1}}(\alpha, i_{j-1})$ in $G(i_{j-1})$ for each $2 \leq j \leq q$. After performing the random matching, add an edge, called a *back* edge, for each canonical path $P_x(\alpha)$ between the start and end vertices in that path. From this, we define a *canonical cycle* $C_x(\alpha)$ to be the cycle formed by the canonical path and the associated back edge. Denote the set of all canonical cycles by $\mathcal{C}$. A few important facts about this graph are noted. First, the length of each canonical cycle is $(3M + 1)q \leq 4M\lambda k^2$. Second, for each accepting configuration α, there are X edge-disjoint canonical cycles associated with α. Finally, the degree of each vertex is at most 3. Figure 3 illustrates this final construction.

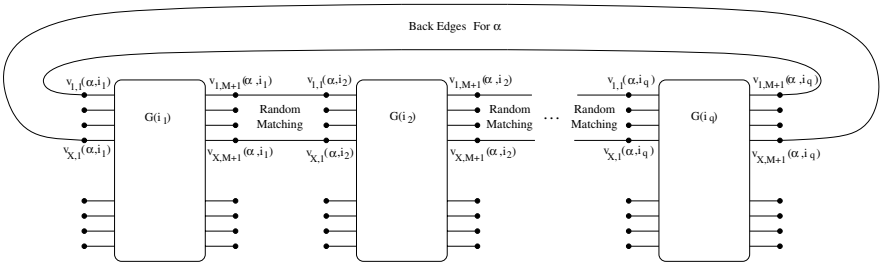


Fig. 3. The final instance for configuration α

We set $X = 2^{2^{2\lambda(k^2+4k)}}$ and $M = 2^{\lambda(k^2+k)}$ in the final construction. Then $X = 2^{\text{polylog}(n)}$ and $M = \text{polylog}(n)$. Each vertex and edge participate in at least one canonical cycle and $|\mathcal{C}| \leq X \cdot 2^{\lambda r} \cdot 2^{\lambda(2k-1)}$ with the length of each cycle in $\mathcal{C}$ being bound by $4M\lambda k^2$. Denoting the number of vertices in the final construction by N we have $N \leq X \cdot 2^{\lambda r} \cdot M \cdot 2^{2\lambda k} \leq X \cdot 2^{O(\log n \log\log n)}$.

3.4 Analysis

Here we show that if Φ is a satisfiable instance of 3SAT then there are many edge-disjoint cycles in the instance we built (those corresponding to the canonical cycles). On the other hand if Φ is a no-instance then the number of edge-disjoint cycles is small. For this part we show that the number of canonical as well as non-canonical cycles is small.

***Φ* is Satisfiable.** If Φ is satisfiable, then there exists a proof Π' for which the probability of acceptance of verifier V is at least $2^{-\lambda}$. For each of the at least $2^{\lambda r - \lambda}$ random strings R that result in verifier V accepting proof Π', choose all of the X canonical cycles corresponding to the configuration $(R, a_1, \dots, a_q)$ where the a_j's, $1 \leq j \leq q$, are the values of the bits read in proof Π' when the

random string is R. It is easy to see that the set of all these canonical cycles are edge-disjoint. Denoting the number of edge-disjoint cycles when Φ is satisfiable by C_{YI}, we have $C_{YI} \geq X \cdot 2^{\lambda r - \lambda} \geq \frac{|\mathcal{C}|}{2^{2\lambda k}}$.

Φ is not satisfiable. Suppose that Φ is not satisfiable and let $\mathcal{C}'$ be a collection of edge-disjoint cycles of the constructed graph G. Define $g = 2^{2\lambda(k^2+k)}$. We say a cycle is short if its length is less than g; otherwise the cycle is called long. Partition $\mathcal{C}'$ into sets $\mathcal{C}_1$, $\mathcal{C}_2$, and $\mathcal{C}_3$ where $\mathcal{C}_1$ is the set of all canonical cycles in $\mathcal{C}'$, $\mathcal{C}_2$ is the set of long non-canonical cycles, and $\mathcal{C}_3$ is the set of short non-canonical cycles. We bound the sizes of each of $\mathcal{C}_1$, $\mathcal{C}_2$, and $\mathcal{C}_3$. The proofs of the following two lemmas are essentially the same as the the corresponding arguments in [9]. We skip repeating them here.

Lemma 4. *If bad event $\mathcal{B}_1$ does not happen, then* $|\mathcal{C}_1| \leq \frac{2C_{YI}}{2^{\lambda k^2 - 2\lambda k - \lambda}}$.

Lemma 5. $|\mathcal{C}_2| \leq \frac{|\mathcal{C}|}{2^{\lambda k^2}} \leq \frac{C_{YI}}{2^{\lambda k^2 - 2\lambda k}}$.

To bound the number of short non-canonical cycles we have to be more careful. For that we first define bad event $\mathcal{B}_2$ as the event $|\mathcal{C}_3| > \frac{C_{YI}}{2^{\lambda k^2}}$.

Lemma 6. *Event $\mathcal{B}_2$ happens with probability at most $\frac{1}{3}$.*

Proof. Let G' be the resultant graph when all of the special edges of G are contracted. An upper bound for the number of cycles of length less than g in G' is clearly an upper bound for the number of cycles of length less than g in G as well. Consider any length $g' < g$ and let us bound the number of non-canonical cycles of length g'. There are two types of edges in G': those that come from random matchings in G and those that are back-edges in G.

Claim. The probability of each edge $e = uv$ appearing in the graph G' given the existence of $g' - 1$ other edges that do not form a canonical path from u to v, is at most $\frac{1}{X - g' + 1}$.

This is easy to see for the case of a non-back-edge (*i.e.* random matching edge) as each matching edge exists with probability at most $\frac{1}{X-g'+1}$ given the existence of $g' - 1$ other edges. The case of a potential back-edge is different as the back-edges are not completely random (each is created between the source and sink of a canonical path; but the path is created randomly). Consider a potential back edge $e = uv$ between a source node u and a sink node v (note that u and v are not necessarily the end points of a canonical path) and suppose we are given the existence of up to $g' - 1$ other edges that do not form a canonical path from u to v. Moreover, consider the partial canonical paths from u and from v using the other at most $g' - 1$ other edges. Since there is currently no canonical path from u to v (otherwise we have a canonical cycle with e), then the probability that u and v are endpoints of the same canonical path is exactly the probability that they will be connected with a new random-matching edge. Thus, the probability that e exists is at most $\frac{1}{X-g'+1}$. Using these arguments, for any potential non-canonical cycle C of length g' the probability that all edges of C exist is at most

$(\frac{1}{X-g'+1})^{g'} \leq (2/X)^{g'}$. A coarse upper bound on the number of potential cycles of length g' in G' is $N^{g'}$ which implies that the expected number of non-canonical cycles of length g' is no more than $\left(\frac{2N}{X}\right)^{g'}$. Summing over all $g' < g$, this yields an upper bound of $\left(\frac{2N}{X}\right)^{g}$ on the expected number of short non-canonical cycles. Since $N \leq X \cdot 2^{\lambda r + 2\lambda k + \lambda(k^2+k)}$, the expected number of cycles of length less than g is at most $2^{\lambda g(r+k^2+4k)} \leq 2^{3\lambda rg}$. By Markov's inequality, the probability that the number of cycles of length less than g is greater than $2^{4\lambda rg} \geq 3 \cdot 2^{3\lambda rg}$ is at most $\frac{1}{3}$. □

Therefore, if event $\mathcal{B}_2$ does not happen, then: $|\mathcal{C}_3| \leq 2^{4\lambda rg} \leq 2^{2^{2\lambda(k^2+3k)+\log\log n}}$, because $r = O(\log n)$. Also, since $\lambda = \beta \log\log n/k^2$ for $\beta >> k^2$, then $\lambda k \geq \log\log n$ resulting in $|\mathcal{C}_3| \leq 2^{2^{2\lambda(k^2+4k)}} \leq X \leq \frac{C_{YI}}{2^{\lambda(r-1)}} \leq \frac{C_{YI}}{2^{\lambda k^2}}$.

Wrap Up. If neither of bad events $\mathcal{B}_1$ nor $\mathcal{B}_2$ happens, then $|\mathcal{C}'| = |\mathcal{C}_1| + |\mathcal{C}_2| + |\mathcal{C}_3| \leq \frac{C_{YI}}{2^{\lambda(k^2-3k)}}$. So the gap between the size of the solution of G for the case that Φ is a yes-instance and for the case that Φ is a no-instance of 3SAT is $\Omega(2^{\lambda(k^2-3k)})$. Remembering that $N \leq X \cdot 2^{\lambda r} \cdot M \cdot 2^{2\lambda k}$, we have $\log N \leq 2^{2\lambda(k^2+4k)} + 3\lambda r$. By selecting β a large constant we have $\log N \leq 2^{2\lambda(k^2+5k)}$ which yields $\sqrt{\log N} \leq 2^{\lambda(k^2-3k)} \cdot 2^{8\lambda k} = \left(2^{\lambda(k^2-3k)}\right)^{1+\frac{8}{k-3}}$. Therefore, $2^{\lambda(k^2-3k)} \geq \log^{\frac{1}{2}-\frac{4}{k+5}} N$ and so for any $\epsilon > 0$, we can choose $k = k(\epsilon) > 0$ such that the gap is at least $\log^{\frac{1}{2}-\epsilon} N$.

The probability of either of events $\mathcal{B}_1$ or $\mathcal{B}_2$ occurring is at most $1/(\text{poly}(n)) + 1/3 \leq 1/2$. So, if a $(\log^{\frac{1}{2}-\epsilon} n)$-approximation algorithm exists for the edge-disjoint cycles problem for any $\epsilon > 0$, then a co-RPTIME($n^{\text{polylog}(n)}$) algorithm for 3SAT exists, which in turn implies the existence of a ZPTIME($n^{\text{polylog}(n)}$) algorithm for 3SAT by a standard result. Thus, for any $\epsilon > 0$, it is hard to approximate the edge-disjoint cycle packing problem within a factor of $\Omega(\log^{\frac{1}{2}-\epsilon} n)$ unless NP $\subseteq$ ZPTIME($n^{\text{polylog}(n)}$).

4 Concluding Remarks

Since each vertex has degree at most 3 in the construction of G, it is easy to see that we get a similar hardness for the vertex-disjoint cycle packing problem. Theorem 2 together with the results of [14,15] yield an almost tight ratio for approximability of EDC in the undirected setting ($O(\sqrt{\log n})$ v.s. $\Omega(\log^{\frac{1}{2}-\epsilon} n)$ for any $\epsilon > 0$). However, the gap between the best approximation ratio and hardness lower bounds for undirected VDC as well as directed EDC (and VDC) are pretty wide.

References

1. Andrews, M., Chuzhoy, J., Khanna, S., Zhang, L.: Hardness of the undirected edge-disjoint paths problem with congestion. In: Proc. of 46th IEEE FOCS, pp. 226–244 (2005)

2. Andrews, M., Zhang, L.: Hardness of the undirected edge-disjoint paths problem. In: Proc. of 37th ACM STOC, pp. 276–283. ACM Press, New York (2005)
3. Balister, P.: Packing digraphs with directed closed trials. Combin. Probab. Comput. 12, 1–15 (2003)
4. Caprara, A., Panconesi, A., Rizzi, R.: Packing cuts in undirected graphs. J. Algorithms 44, 1–11 (1999)
5. Caprara, A., Panconesi, A., Rizzi, R.: Packing cycles in undirected graphs. J. Algorithms 48, 239–256 (2003)
6. Chekuri, C., Khanna, S.: Edge disjoint paths revisited. In: Proc. of 14th ACM-SIAM SODA, pp. 628–637 (2003)
7. Chekuri, C., Khanna, S., Shepherd, B.: An $O(\sqrt{n})$ Approximation and Integrality Gap for Disjoint Paths and UFP. Theory of Computing 2, 137–146 (2006)
8. Chuzhoy, J., Guha, S., Halperin, E., Khanna, S., Kortsarz, G., Krauthgamer, R., Naor, S.: Tight lower bounds for the asymmetric k-center problem. Journal of ACM 52(4), 538–551 (2005)
9. Chuzhoy, J., Khanna, S.: New hardness results for undirected edge disjoint paths, Manuscript (2005)
10. Dor, D., Tarsi, M.: Graph decomposition is NPC – A complete proof of Holyer's conjecture. In: Proc. of 20th ACM STOC, pp. 252–263. ACM Press, New York (1992)
11. Guruswami, V., Khanna, S., Rajaraman, R., Shepherd, B., Yannakakis, M.: Near-Optimal Hardness Results and Approximation Algorithms for Edge-Disjoint Paths and Related Problems. J. of Computer and System Sciences 67(3), 473–496 (2003) Earlier version in STOC 1999.
12. Halldorsson, M.M., Kortsarz, G., Radhakrishnan, J., Sivasubramanian, S.: Complete Partitions of Graphs. Combinatorica (to appear)
13. Kleinberg, J.: Approximation algorithms for disjoint paths problems, PhD. Thesis, MIT, Cambridge, MA (May 1996)
14. Krivelevich, M., Nutov, Z., Yuster, R.: Approximation algorithms for cycle packing problems. In: Proc. of 16th ACM-SIAM SODA, pp. 556–561 (2005)
15. Krivelevich, M., Nutov, Z., Salavatipour, M.R., Verstraete, J., Yuster, R.: Approximation Algorithms and Hardness Results for Cycle Packing Problems. ACM Transactions on Algorithms (to appear)
16. Salavatipour, M.R., Verstraete, J.: Disjoint cycles: Integrality gap, hardness, and approximation. In: Jünger, M., Kaibel, V. (eds.) Integer Programming and Combinatorial Optimization. LNCS, vol. 3509, pp. 51–65. Springer, Heidelberg (2005)
17. Samorodnitsky, A., Trevisan, L.: A PCP characterization of NP with optimal amortized query complexity. In: Proc. of 32nd ACM STOC, pp. 191–199. ACM Press, New York (2000)
18. Seymour, P.D.: Packing directed circuits fractionally. Combinatorica 15, 281–288 (1995)
19. Varadarajan, K.R., Venkataraman, G.: Graph decomposition and a greedy algorithm for edge-disjoint paths. In: Proc. of 15 ACM-SIAM SODA, pp. 379–380 (2004)

Succinct Representation of Labeled Graphs*

Jérémy Barbay[1], Luca Castelli Aleardi[2], Meng He[1,3], and J. Ian Munro[1]

[1] Cheriton School of Computer Science, University of Waterloo, Canada
{jbarbay,mhe,imunro}@uwaterloo.ca
[2] LIX (Ecole Polytechnique, France) and
CS Department (Université Libre de Bruxelles, Belgium)
amturing@lix.polytechnique.fr
[3] School of Computer Science, Carleton University, Canada
mhe@cg.scs.carleton.ca

Abstract. In many applications, the properties of an object being modeled are stored as labels on vertices or edges of a graph. In this paper, we consider succinct representation of labeled graphs. Our main results are the succinct representations of labeled and multi-labeled graphs (we consider vertex labeled planar triangulations, as well as edge labeled planar graphs and the more general k-page graphs) to support various label queries efficiently. The additional space cost to store the labels is essentially the information-theoretic minimum. As far as we know, our representations are the first succinct representations of labeled graphs. We also have two preliminary results to achieve the main results. First, we design a succinct representation of unlabeled planar triangulations to support the rank/select of edges in ccw (counter clockwise) order in addition to the other operations supported in previous work. Second, we design a succinct representation for a k-page graph when k is large to support various navigational operations more efficiently. In particular, we can test the adjacency of two vertices in $O(\lg k \lg\lg k)$ time, while previous work uses $O(k)$ time (10; 14).

1 Introduction

Graphs are fundamental combinatorial objects, widely used to represent various types of data. As modern applications often process large graphs, the problem of designing space-efficient data structures to represent graphs has attracted a great deal of attention. In particular the idea of *succinct data structures*, i.e. data structures that occupy space close to the information-theoretic lower bound while supporting efficient navigational operations, has been applied to various classes of graphs (5; 6; 7; 8; 12; 14).

Previous work focused on succinct graph representations which support efficiently testing the adjacency between two vertices and listing the edges incident

* This work was supported by NSERC of Canada, the Canada Research Chairs program, the IGM (University of Marne-la-Vallée) and the French "ACI Masses de données" program via the Geocomp project.

T. Tokuyama (Ed.): ISAAC 2007, LNCS 4835, pp. 316–328, 2007.

to a vertex (5; 6; 14). However, in many applications, such connectivity information is associated with labels on the edges or vertices of the graph, and the space required to encode those labels dominates the space used to encode the connectivity information, even when the encoding of the labels is compressed (11). For example, when surface meshes are associated with properties such as color and texture information, more bits per vertex are required to encode those labels than to encode the graph itself. We address this problem by designing succinct representations of *labeled graphs*, where labels from alphabet $[\sigma]$ [1] are associated with edges or vertices. These representations efficiently support label-based connectivity queries, such as retrieving the neighbors associated with a given label. Our results are under the word RAM model with word size $\Theta(\lg n)$ bits.[2].

We investigate three important classes of graphs: planar triangulations, planar graphs and k-page graphs. Planar graphs, in particular planar triangulations, correspond to the connectivity information underlying surface meshes. Triangle meshes are one of the most fundamental representations for geometric objects: in computational geometry they are one natural way to represent surface models, and in computer graphics triangles are the basic geometric primitive (for efficient rendering). k-page graphs have applications in several areas, such as sorting with parallel stacks (17), fault-tolerant processor arrays (15) and VLSI (9).

2 Preliminaries

2.1 Related Work

Jacobson (12) first proposed to represent unlabeled graphs succinctly. His approach is based on the concept of *book embedding* (4). A *k-page embedding* is a topological embedding of a graph with the vertices along the spine and edges distributed across k pages, each of which is an outerplanar graph. The minimum number of pages, k, for a particular graph has been called the *pagenumber* or *book thickness*. Jacobson showed how to represent a k-page graph using $O(kn)$ bits to support adjacency tests in $O(\lg n)$ bit probes, and listing the neighbors of a vertex in $O(d \lg n + k)$ bit probes, where d is the vertex degree.

Munro and Raman (14) improved his results under the word RAM model by showing how to represent a graph using $2kn + 2m + o(kn + m)$ bits to support adjacency tests and the computation of the degree of a vertex in $O(k)$ time, and the listing of all the neighbors of a given vertex in $O(d + k)$ time. Gavoille and Hanusse (10) proposed a different tradeoff. They proposed an encoding in $2(m+i)\lg k+4(m+i)+o(km)$ bits, where i is the number of isolated vertices, to support the adjacency test in $O(k)$ time. As any planar graph can be embedded in at most 4 pages (18), these results can be applied directly to planar graphs. In particular, a planar graph can be represented using $8n + 2m + o(n)$ bits to

[1] We use $[\sigma]$ to denote the set $\{1, 2, \ldots, \sigma\}$ of references to arbitrary labels, as indeed the alphabet of labels.

[2] We use $\log_2 x$ to denote the logarithmic base 2 and $\lg x$ to denote $\lceil \log_2 x \rceil$. Occasionally this matters.

support adjacency tests and the computation of the degree of a vertex in $O(1)$ time, and the listing of all the neighbors of a given vertex in $O(d)$ time (14).

A different line of research is based on the canonical ordering of planar graphs. Chuang *et al.* (8) designed a succinct representation of planar graphs of n vertices and m edges in $2m+(5+\epsilon)n+o(m+n)$ bits, for any constant $\epsilon > 0$, to support the operations on planar graphs in asymptotically the same amount of time as the approach described in the previous paragraph. Chiang *et al.* (7) further reduced the space cost to $2m + 3n + o(m + n)$ bits. When a planar graph is triangulated, Chuang *et al.* (8) showed how to represent it using $2m + 2n + o(m + n)$ bits.

Based on a partition algorithm, Castelli Aleardi *et al.* (5) proposed a succinct representation of planar triangulations with a boundary. Their data structure uses 2.175 bits per triangle to support various operations efficiently. Castelli Aleardi *et al.* (6) further extended this approach to design succinct representations of 3-connected planar graphs and triangulations using 2 bits per edge and 1.62 bits per triangle respectively, which asymptotically match the respective entropy of these two types of graphs.

2.2 Multiple Parentheses

Chuang *et al.* (8) proposed succinct representation of *multiple parentheses*, a string of $O(1)$ types that may be unbalanced. Thus a multiple parenthesis sequence of p types of parentheses is a sequence over the alphabet $\{'(_1', '(_2', ..., '(_p', ')'_1, ')'_2, ..., ')'_p\}$. We call $'(_i'$ and $')'_i$ *type-i opening parenthesis* and *type-i closing parenthesis*, respectively. The operations considered are:

- $\texttt{m\_rank}(S, i, \alpha)$: the number of parentheses α in $S[1..i]$;
- $\texttt{m\_select}(S, i, \alpha)$: the position of the i^{th} parenthesis α;
- $\texttt{m\_first}_\alpha(S, i)$ ($\texttt{m\_last}_\alpha(S, i)$): the position of the first (last) parenthesis α after (before) $S[i]$;
- $\texttt{m\_match}(S, i)$: the position of the parenthesis matching $S[i]$;
- $\texttt{m\_enclose}_k(S, i_1, i_2)$: the position of the closest matching parenthesis pair of type k which encloses $S[i_1]$ and $S[i_2]$.

Chuang *et al.* (8) showed how to construct a $o(|S|)$-bit auxiliary data structure, for a string S of $O(1)$ types of parentheses stored explicitly, to support the above operations in constant time. We show how to improve this result in Corollary 1, and propose an encoding for the case when the number of types of parentheses is non-constant in Theorem 3.

2.3 Succinct Indexes for Binary Relations

Barbay *et al.* (2) showed how to achieve data abstraction in succinct data structures by designing *succinct indexes*. Given an abstract data type (ADT) to access the given data, the goal is to design auxiliary data structures (i.e. succinct indexes) that occupy asymptotically less space than the information-theoretic lower bound on the space required to encode the given data, and support an extended set of operations using the basic operators defined in the ADT.

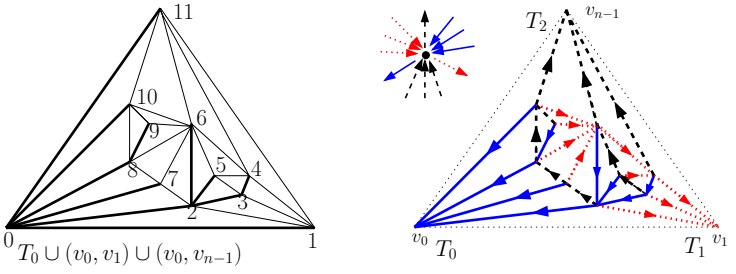


Fig. 1. A triangulated planar graph of 12 vertices with its canonical spanning tree $\overline{T}_0$ (on the left). On the right, it shows the triangulation induced with a realizer, as well as the local condition.

They considered sequences of n objects where each object can be associated with a subset of labels from $[\sigma]$, this association being defined by a binary relation of t pairs from $[n]\times[\sigma]$. The operations include: `object_access`(x, i), the i^{th} label associated with x in lexicographic order; `label_rank`(α, x), the number of objects labeled α up to (and including) x; `label_select`(α, r), the position of the r^{th} object labeled α; and `label_access`(x, α), whether object x is associated with label α. They defined the ADT through `object_access` and designed a succinct index of $t \cdot o(\lg \sigma)$ bits to support other operators efficiently.

2.4 Realizers and Planar Triangulations

A key notion in this paper is that of realizers of planar triangulations (see Figure 1 for an example).

Definition 1 (Schnyder (16)). *A **realizer** of a planar triangulation $\mathcal{T}$ is a partition of the set of the internal edges into three sets T_0, T_1 and T_2 of directed edges, such that for each internal vertex v the following conditions hold:*

- *v has exactly one outgoing edge in each of the three sets T_0, T_1 and T_2;*
- ***local condition:** the edges incident to v in ccw order are: one outgoing edge in T_0, zero or more incoming edges in T_2, one outgoing edge in T_1, zero or more incoming edges in T_0, one outgoing edge in T_2, and finally zero or more incoming edges in T_1.*

A fundamental property of realizers that we use extensively in Section 3 is:

Lemma 1 (Schnyder (16)). *Consider a planar triangulation $\mathcal{T}$ of n vertices, with exterior face (v_0, v_1, v_{n-1}). Then $\mathcal{T}$ always admits a realizer $R = (T_0, T_1, T_2)$ and each set of edges in T_i is a spanning tree of all internal vertices. More precisely, T_0, T_1 and T_2 are spanning trees of $\mathcal{T} \setminus \{v_1, v_{n-1}\}$, $\mathcal{T} \setminus \{v_0, v_{n-1}\}$ and $\mathcal{T} \setminus \{v_0, v_1\}$, respectively.*

3 Vertex Labeled Planar Triangulations

3.1 Three New Traversal Orders on a Planar Triangulation

A key notion in the development of our results is that of three new traversal orders of planar triangulations based on realizers. Let $\mathcal{T}$ be a planar triangulation of n vertices and m edges, with exterior face (v_0, v_1, v_{n-1}). We denote its realizer by (T_0, T_1, T_2) following Lemma 1. By Lemma 1, T_0, T_1 and T_2 are three spanning trees of the internal nodes of $\mathcal{T}$, rooted at v_0, v_1 and v_{n-1}, respectively. We add the edges (v_0, v_1) and (v_0, v_{n-1}) to T_0, and call the resulting tree, $\overline{T}_0$, the *canonical spanning tree* of $\mathcal{T}$ (8). In this section, we denote each vertex by its number in *canonical ordering*, which is the ccw preorder number in $\overline{T}_0$.

Definition 2. ***The zeroth order*** π_0 *is defined on all the vertices of* $\mathcal{T}$ *and is simply given by the preorder traversal of* $\overline{T_0}$ *starting at* v_0 *in* counter clockwise order *(ccw order).*

The first order π_1 *is defined on the vertices of* $\mathcal{T} \setminus v_0$ *and corresponds to a traversal of the edges of* T_1 *as follows. Perform a preorder traversal of the contour of* $\overline{T_0}$ *in a ccw manner. During this traversal, when visiting a vertex* v*, we enumerate consecutively its incident edges* $(v, u_1), \ldots, (v, u_i)$ *in* T_1*, where* v *appears before* u_i *in* π_0*. The traversal of the edges of* T_1 *naturally induces an order on the nodes of* T_1*: each node (different from* v_1*) is uniquely associated with its parent edge in* T_1*.*

The second order π_2 *is defined on the vertices of* $\mathcal{T} \setminus \{v_0, v_1\}$ *and can be computed in a similar manner by performing a preorder traversal of* T_0 *in* clockwise order *(cw order). When visiting in cw order the contour of* $\overline{T_0}$*, the edges in* T_2 *incident to a node* v *are listed consecutively to induce an order on the vertices of* T_2*.*

Note that the orders π_1 and π_2 do not correspond to previously studied traversal orders on the trees T_1 and T_2, as they are dependent on $\overline{T_0}$ through π_0 (see Figure 2). The following lemma is crucial (we omit the proof):

Lemma 2. *For any node* x*, its children in* T_1 *(or* T_2*), listed in ccw order (or cw order), have consecutive numbers in* π_1 *(or* π_2*). In the case of* $\overline{T_0}$*, the children of* x *are listed consecutively by a* `DFUDS` *(or* Depth First Unary Degree Sequence *(3)) traversal of* $\overline{T_0}$*.*

3.2 Representing Planar Triangulations

We consider the following operations on unlabeled planar triangulations:

- `adjacency`(x, y), whether vertices x and y are adjacent;
- `degree`(x), the degree of vertex x;
- `select_neighbor_ccw`(x, y, r), the r^{th} neighbor of vertex x starting from vertex y in ccw order if x and y are adjacent, and ∞ otherwise;
- `rank_neighbor_ccw`(x, y, z), the number of neighbors of vertex x between (and including) the vertices y and z in ccw order if y and z are both neighbors of x, and ∞ otherwise.

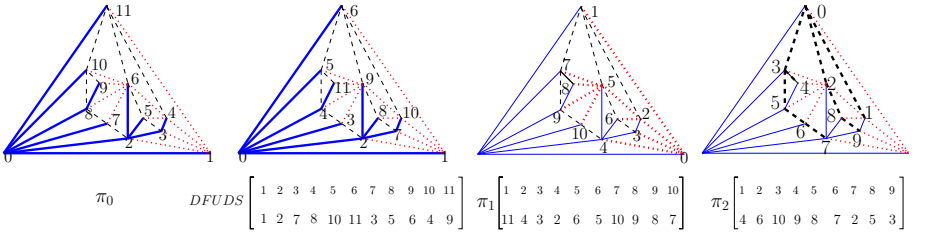


Fig. 2. A planar triangulation induced with one realizer. The three orders π_0, π_1 and π_2, as well as the order induced by a DFUDS traversal of $\overline{T_0}$ are also shown.

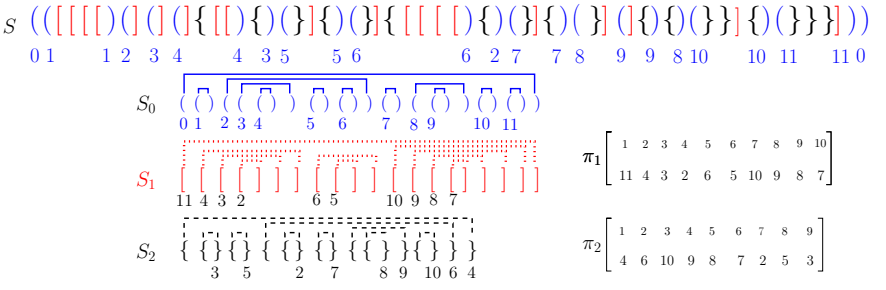


Fig. 3. The multiple parenthesis string encoding of the triangulation in Figure 2

- $\Pi_j(i)$, given the number of a node v_i in π_0 it returns the number of v_i in π_j;
- $\Pi_j^{-1}(i)$, given the number of a node v_i in π_j it returns its rank in π_0.

To represent a planar triangulation $\mathcal{T}$, we compute a realizer (T_0, T_1, T_2) of $\mathcal{T}$ following Lemma 1. We then encode the three trees T_0, T_1 and T_2 using a multiple parenthesis sequence S of length $2m$ consisting of three types of parenthesis. S is obtained by performing a preorder traversal of the canonical spanning tree $\overline{T}_0 = T_0 \cup (v_0, v_1) \cup (v_0, v_{n-1})$ and using different types of parentheses to describe the edges of $\overline{T}_0$, T_1 and T_2. We use parentheses of the first type, namely $'('$ and $')'$, to encode the tree $\overline{T}_0$, and other types of parentheses, $'['$, $']'$, $'\{'$, $'\}'$, to encode the edges of T_1 and T_2. We use S_0, S_1 and S_2 to denote the subsequences of S that contain all the first, second, and the third types of parentheses, respectively. We construct S as follows (see Figure 3 for an example).

Let $v_0, \ldots, v_{n-1}$ be the ccw preorder of the vertices of $\overline{T}_0$. Then the string S_0 is simply the balanced parenthesis encoding of the tree $\overline{T}_0$ (14): S_0 can be obtained by performing a ccw preorder traversal of the contour of $\overline{T}_0$, writing down an opening parenthesis when an edge of $\overline{T}_0$ is traversed for the first time, and a closing parenthesis when it is visited for the second time. During the traversal of $\overline{T}_0$, we insert in S a pair of parentheses $'['$ and $']'$ for each edge of T_1, and a pair of parentheses $'\{'$ and $'\}'$ for each edge in T_2. More precisely, when visiting in ccw order the edges incident to a vertex v_i, we insert:

- A $'['$ for each edge (v_i, v_j) in T_1, where $i < j$, before the parenthesis $')'$ corresponding to v_i;

- A $']'$ for each edge (v_i, v_j) in T_1, where $i < j$, after the parenthesis $'('$ corresponding to v_j;
- A $'\}'$ for each edge (v_i, v_j) in T_2, where $i > j$, after the parenthesis $'('$ corresponding to v_i;
- A $'\{'$ for each edge (v_i, v_j) in T_2, where $i > j$, before the parenthesis $')'$ corresponding to v_j.

Thus S is of length $2m$, consisting of three types of parenthesis. It is easy to observe that the subsequences S_1 and S_2 are balanced parenthesis sequences of length $2(n-1)$ and $2(n-2)$, respectively.

We first observe some basic properties of the string S. Recall that a node v_i can be referred to by its preorder number in T_0, and by the position of the matching parenthesis pair $(_i$ and $)_i$ (let p_i and q_i denote their positions in S). Let be p_f (or q_l) be the position of the opening (or closing) parenthesis in S corresponding to the first (or last) child of node v_i in T_0.

Property 1. The following basic facts hold:

- Two nodes v_i and v_j are adjacent if and only if there is one common incident edge (v_i, v_j) in exactly one of the trees T_0, T_1 or T_2;
- $p_i < p_f < q_l < q_i$;
- The number of edges incident to v_i and not belonging to the tree T_0 is $(p_f - p_i - 1) + (q_i - q_l - 1)$;
- If v_i is not a leaf in $\overline{T_0}$, between the occurrences of the $'('$ that correspond to the vertices v_i and v_{i+1} (note that the $'('$ corresponding to v_{i+1} is at position p_f), there is exactly one $']'$. Similarly, there is exactly one $'\{'$ between the $')'$ that correspond to the vertices v_i and the $')'$ at position q_l.

Observe that S_0 is the balanced parenthesis encoding of the tree $\overline{T_0}$ (14), so that if we store S_0 and construct the auxiliary data structures for S_0 as in (14), we can support a set of navigational operators on $\overline{T_0}$. S can be represented using the approach of Chuang *et al.* (8) (see Section 2.2) in $2m \lg 6 + o(m) = 2m\lceil \log_2 6 \rceil + o(m) = 6m + o(m)$ bits. However, this encoding does not support the computation of an arbitrary word in S_0, so that we cannot navigate in the tree $\overline{T_0}$ without storing S_0 explicitly, which will cost essentially 2 additional bits per node. To reduce this space redundancy, and to decrease the item $2m\lceil \log_2 6 \rceil$ to $2m \log_2 6 + o(m)$, we have the following lemma (we omit the proof):

Lemma 3. *The string S can be stored in $2m \log_2 6 + o(m)$ bits to support the operators listed in Section 2.2 in constant time, as well as the computation of an arbitrary word, or $\Theta(n)$ bits of the balanced parenthesis sequence of $\overline{T_0}$.*

The same approach can be directly applied to a sequence of $O(1)$ types of parentheses:

Corollary 1. *Consider a multiple parenthesis sequence M of $2n$ parenthesis of p types, where $p = O(1)$. M can be stored using $2n \log(2p) + o(n)$ bits to support in $O(1)$ time the operators listed in Section 2.2, as well as the computation of an arbitrary word, or $\Theta(n)$ bits of the balanced parenthesis sequence of the parentheses of a given type in M.*

The following theorem shows how to support the navigational operations on triangulations. While the space used here is a little more than that of (7), the explicit use of the three parenthesis sequences seems crucial to exploiting the realizers to provide an efficient implementation supporting $\Pi_j(i)$ and $\Pi_j^{-1}(i)$.

Theorem 1. *A planar triangulation $\mathcal{T}$ of n vertices and m edges can be represented using $2m \log_2 6 + o(m)$ bits to support* `adjacency`, `degree`, `select_neighbor_ccw`, `rank_neighbor_ccw` *as well as the $\Pi_j(i)$ and $\Pi_j^{-1}(i)$ operators (for $j \in \{1, 2\}$) in $O(1)$ time.*

Proof. We construct the string S for $\mathcal{T}$ as shown in this section, and store it using $2m \log_2 6 + o(m)$ bits by Lemma 3. Recall that S_0 is the balanced parenthesis encoding of $\overline{T_0}$, and that we can compute an arbitrary word of S_0 from S. Thus we can construct additional auxiliary structures using $o(n) = o(m)$ bits (13; 14) to support the navigational operations on $\overline{T_0}$. As each vertex is denoted by its number in canonical ordering, vertex x corresponds to the x^{th} opening parenthesis in S_0. We now show that these structures are sufficient.

To compute `adjacency`(x, y), recall that x and y are adjacent iff one is the parent of the other in one of the trees $\overline{T_0}$, T_1 and T_2. As S_0 encodes the balanced parenthesis sequence of $\overline{T_0}$, we can trivially check whether x (or y) is the parent of y (or x) using existing algorithms on S_0 (14). To test adjacency in T_1, we recall that x is the parent of y iff the (only) outgoing edge of y, denoted by a $']'$, is an incoming edge of x, denoted by a $'['$. It then suffices to retrieve the first $']'$ after the y^{th} $'('$ in S, given by `m_first`$_{']'}(S,$`m_select`$(S, y, '('))$, and compute the index, i, of its matching closing parenthesis, $'['$, in S. We then check whether the nearest succeeding closing parenthesis $')'$ of the $'['$ retrieved, located using `m_first`$_{')'}(S, i)$, matches the x^{th} opening parenthesis $'('$ in S. If it does, then x is the parent of y in T_1. We use a similar approach to test the adjacency in T_2.

To compute `degree`(x), let d_0, d_1 and d_2 be the degrees of x in the trees $\overline{T_0}$, T_1 and T_2 (we denote the degree of a node in a tree as the number of nodes adjacent to it), respectively, so that the sum of these three values is the answer. To compute d_0, we use S_0 and the algorithm to compute the degree of a node in an ordinal tree using its balanced parenthesis representation by Chuang *et al.* (8). To compute $d_1 + d_2$, if x has children in $\overline{T_0}$, we first compute the indices, i_1 and i_2, of the x^{th} and the $x + 1^{\text{th}}$ $'('$ in S, and the indices, j_1 and j_2, of the $(n - x)^{\text{th}}$ and the $(n - x + 1)^{\text{th}}$ $')'$ in S in constant time. By the third item of Property 1, we have the property $d_1 + d_2 = (i_2 - i_1 - 1) + (j_2 - j_1 - 1)$. The case when x is a leaf in $\overline{T_0}$ can be handled similarly.

To support `select_neighbor_ccw` and `rank_neighbor_ccw`, we make use of the local condition of realizers in Definition 1. The local condition tells us that, given a vertex x, its neighbors, when listed in ccw order, form the following six types of vertices: x's parent in $\overline{T_0}$, x's children in T_2, x's parent in T_1, x's children in $\overline{T_0}$, x's parent in T_2, and x's children in T_1. The i^{th} child of x in ccw order in $\overline{T_0}$ can be computed in constant time, and the number of siblings before a given child of x in ccw order can also be computed in constant time using the algorithms of Lu and Yeh (13). The children of x in T_1 corresponds to the

parentheses $'['$ between the $(n-x)^{\text{th}}$ and the $(n-x+1)^{\text{th}}$ $')'$ in S, and because of the construction of S, if u and v are both children of x, and u occurs before v in π_1, then u is also before v in ccw order among x's children. The children of x in T_2 have a similar property. Thus the operators supported on S allow us to perform rank/select on x's children in T_1 and T_2 in ccw order. As we can also compute the number of each type of neighbors of x in constant time, this allows us to support `select_neighbor_ccw` and `rank_neighbor_ccw` in $O(1)$ time.

To compute $\Pi_1(i)$, we first locate the position, j, of the i^{th} $'('$ in S, which is $\texttt{m\_select}(S, i, '(')$. We then locate the position, k, of the first $']'$ after position j, which is $\texttt{m\_first}_{']'}(S, j)$. After that, we locate the matching parenthesis of $S[j]$ using $\texttt{m\_match}(S, j)$ (p denotes the result). $S[p]$ is the parenthesis $'['$ that corresponds to the edge between u_i and its parent in T_1, and by the construction algorithm of S, the rank of $S[p]$ is the answer, which is $\texttt{m\_rank}(S, p, '[')$. The computation of Π_1^{-1} is exactly the inverse of the above process. Π_2 and Π_2^{-1} can be supported similarly. □

3.3 Vertex Labeled Planar Triangulations

In addition to unlabeled operators, we present a set of operators that allow efficient navigation in a labeled graph (these are natural extensions to navigational operators on labeled trees):

- `lab_degree`(α, x), the number of the neighbors of vertex x in G labeled α;
- `lab_select_ccw`(α, x, y, r), the r^{th} vertex labeled α among neighbors of vertex x after vertex y in ccw order, if y is a neighbor of x, and ∞ otherwise;
- `lab_rank_ccw`(α, x, y, z), the number of the neighbors of vertex x labeled α between y and z in ccw order if y and z are neighbors of x, and ∞ otherwise.

We define the interface of the ADT of labeled planar triangulations through `node_label`(v, i), which returns the i^{th} label associated to vertex v (i.e. the v^{th} vertex in canonical ordering).

Recall that Lemma 3 encodes the string S constructed in Section 3.2 to support the computation of an arbitrary word of S_0, which is the balanced parenthesis sequence of the tree $\overline{T_0}$. In this section, we consider the `DFUDS` sequence of $\overline{T_0}$. We have the following lemma (we omit the proof).

Lemma 4. *The string S can be stored in $(2\log_2 6 + \epsilon)m + o(m)$ bits, for any ϵ such that $0 < \epsilon < 1$, to support in $O(1)$ time the operators listed in Section 2.2, as well as the computation of an arbitrary word, or $\Theta(n)$ bits of the balanced parenthesis sequence, and of the* `DFUDS` *sequence of $\overline{T_0}$.*

As Barbay *et al.* (2) did for multi-labeled trees, we now construct succinct indexes for vertex labeled planar triangulations. The main idea is to combine our succinct representation of planar triangulations with three instances of the succinct indexes for related binary relations:

Theorem 2. *Consider a multi-labeled planar triangulation $\mathcal{T}$ of n vertices, associated with σ labels in t pairs ($t \geq n$). Given the support of* `node_label` *in*

$f(n, \sigma, t)$ time on the vertices of $\mathcal{T}$, there is a succinct index using $t \cdot o(\lg \sigma)$ bits which supports `lab_degree`, `lab_select_ccw` *and* `lab_rank_ccw` *in $O((\lg\lg\lg\sigma)^2(f(n,\sigma,t) + \lg\lg\sigma))$ time.*

To design a succinct representation of multi-labeled graphs using the above theorem, we use the approach of Barbay *et al.* (1) to encode R_0 using $t \lg \sigma + O(t)$ bits to support `object_access` in constant time, which directly supports `node_label` in $O(1)$ time. Thus:

Corollary 2. *A multi-labeled planar triangulation $\mathcal{T}$ of n vertices, associated with σ labels in t pairs ($t \geq n$) can be represented using $t \lg \sigma + t \cdot o(\lg \sigma)$ bits to support* `node_label` *in $O(1)$ time, and* `lab_degree`, `lab_select_ccw` *and* `lab_rank_ccw` *in $O((\lg\lg\lg\sigma)^2 \lg\lg\sigma)$ time.*

4 Edge Labeled Graphs with Pagenumber k

4.1 Multiple Parentheses

We now consider the succinct representation of multiple parenthesis sequences of p types of parentheses, where p is not a constant. We consider the following operations on a multiple parenthesis sequence $S[1..2n]$ in addition to those defined in Section 2.2: `m_rank'`(S, i), the rank of the parenthesis at position i among parentheses of the same type in S; `m_findopen`(S, i) (`m_findclose`(S, i)), the matching closing (opening) parenthesis of the same type for the opening (closing) parenthesis at position i in S. Note that `m_findopen` and `m_findclose` are identical to the operator `m_match`. We define them here for the simplicity of the proofs of the theorems in this section. We have the following theorem (we omit all the proofs in this section because of space constraint):

Theorem 3. *A multiple parenthesis sequence of $2n$ parentheses of p types, in which the parentheses of any given type are balanced, can be represented using $2n \lg p + o(n \lg p)$ bits to support* `m_access`, `m_rank'`, `m_findopen` *and* `m_findclose` *in $O(\lg\lg p)$ time, and* `m_select` *in $O(1)$ time. Alternatively, $(2 + \epsilon)n \lg p + o(n \lg p)$ bits are sufficient to support these operations in $O(1)$ time, for any constant ϵ such that $0 < \epsilon < 1$.*

4.2 Graphs with Pagenumber k for Large k

In this section, on unlabeled graphs with page number k, we consider the operators `adjacency` and `degree` defined in Section 3.2, and the operator `neighbors`(x), returning the neighbors of x.

Previous results on succinctly representing k-page graphs (10; 14) support `adjacency` in $O(k)$ time. The lower-order term in the space cost of the result of Gavoille and Hanusse (10) is $o(km)$, which is dominant when k is large. Thus previous results mainly deal with the case when k is small. We consider large k.

Theorem 4. *A k-page graph of n vertices and m edges can be represented using $n + 2m \lg k + o(m \lg k)$ bits to support* `adjacency` *in $O(\lg k \lg \lg k)$ time,* `degree` *in $O(1)$ time, and* `neighbors`(x) *in $O(d(x) \lg \lg k)$ time where $d(x)$ is the degree of x. Alternatively, it can be represented in $n + (2 + \epsilon)m \lg k + o(m \lg k)$ bits to support* `adjacency` *in $O(\lg k)$ time,* `degree` *in $O(1)$ time, and* `neighbors`(x) *in $O(d(x))$ time, for any constant ϵ such that $0 < \epsilon < 1$.*

4.3 Edge Labeled Graphs with Pagenumber k

We consider the following operations on edge labeled graphs:
- `lab_adjacency`(α, x, y), whether there is an edge labeled α between vertices x and y;
- `lab_degree_edge`(α, x), the number of edges incident to vertex x that are labeled α;
- `lab_edges`(α, x), the edges incident to vertex x that are labeled α.

We first design succinct representation of edge labeled graphs with one page:

Lemma 5. *An outerplanar graph of n vertices and m edges in which the edges are associated with σ labels in t pairs ($t \geq n$) can be represented using $n + t(\lg \sigma + o(\lg \sigma))$ bits to support* `lab_adjacency` *and* `lab_degree_edge` *in $O(\lg \lg \sigma (\lg \lg \lg \sigma)^2)$ time, and* `lab_edges`(α, x) *in $O($`lab_degree_edge`$(\alpha, x) \lg \lg \sigma \lg \lg \lg \sigma)$ time.*

To support an edge labeled graph with k pages, we can use Lemma 5 to represent each page and combine all the pages to support navigational operations. Alternatively, we can use Theorem 4 and a similar approach to Lemma 5 to achieve a different tradeoff to improve the time efficiency for large k.

Theorem 5. *A k-page graph of n vertices and m edges in which the edges are associated with σ labels in t pairs ($t \geq n$) can be represented using $kn + t(\lg \sigma + o(\lg \sigma))$ bits to support* `lab_adjacency` *and* `lab_degree_edge` *in $O(k \lg \lg \sigma (\lg \lg \lg \sigma)^2)$ time, and* `lab_edges`(α, x) *in $O($`lab_degree_edge`$(\alpha, x) \lg \lg \sigma \lg \lg \lg \sigma + k)$ time. Alternatively, it can be represented using $n + (2m + \epsilon) \lg k + o(m \lg k) + m(\lg \sigma + o(\lg \sigma))$ bits to support* `lab_adjacency` *in $O(\lg k \lg \lg \sigma (\lg \lg \lg \sigma)^2)$ time,* `lab_degree_edge` *in $O(\lg \lg \sigma (\lg \lg \lg \sigma)^2)$ time, and* `lab_edges`(α, x) *in $O($`lab_degree_edge`$(\alpha, x) \lg \lg \sigma \lg \lg \lg \sigma)$ time, for any constant ϵ such that $0 < \epsilon < 1$.*

Corollary 3. *An edge-labeled planar graph of n vertices and m edges in which the edges are associated with σ labels in t pairs ($t \geq n$) can be represented using $4n + t(\lg \sigma + o(\lg \sigma))$ bits to support* `lab_adjacency` *and* `lab_degree_edge` *in $O(\lg \lg \sigma (\lg \lg \lg \sigma)^2)$ time, and* `lab_edges`(α, x) *in $O($`lab_degree_edge`$(\alpha, x) \lg \lg \sigma \lg \lg \lg \sigma)$ time.*

5 Concluding Remarks

In this paper, we present a framework of succinctly representing the properties of graphs in the form of labels. We expect that our approach can be extended to support other types of planar graphs, which is an open research topic. Another open problem is to represent vertex labeled k-page graphs succinctly.

Our final comment is that because Theorem 2 provides a succinct index for vertex labeled planar triangulations, we can in fact store the labels in compressed form as Barbay *et al.* (2) have done to compress strings, binary relations and multi-labeled trees, while still supporting the same operations. This also applies to Theorem 5, where we apply succinct indexes for binary relations.

References

[1] Barbay, J., Golynski, A., Munro, J.I., Rao, S.S.: Adaptive searching in succinctly encoded binary relations and tree-structured documents. In: Lewenstein, M., Valiente, G. (eds.) CPM 2006. LNCS, vol. 4009, pp. 24–35. Springer, Heidelberg (2006)

[2] Barbay, J., He, M., Munro, J.I., Rao, S.S.: Succinct indexes for strings, binary relations and multi-labeled trees. In: Proceedings of the 18th Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 680–689. ACM Press, New York (2007)

[3] Benoit, D., Demaine, E.D., Munro, J.I., Raman, R., Raman, V., Rao, S.S.: Representing trees of higher degree. Algorithmica 43(4), 275–292 (2005)

[4] Bernhart, F., Kainen, P.C.: The book thickness of a graph. Journal of Combinatorial Theory, Series B 27(3), 320–331 (1979)

[5] Castelli-Aleardi, L., Devillers, O., Schaeffer, G.: Succinct representation of triangulations with a boundary. In: Dehne, F., López-Ortiz, A., Sack, J.-R. (eds.) WADS 2005. LNCS, vol. 3608, pp. 134–145. Springer, Heidelberg (2005)

[6] Castelli-Aleardi, L., Devillers, O., Schaeffer, G.: Optimal succinct representations of planar maps. In: Proceedings of the 22nd ACM Annual Symposium on Computational Geometry, pp. 309–318 (2006)

[7] Chiang, Y.-T., Lin, C.-C., Lu, H.-I.: Orderly spanning trees with applications to graph encoding and graph drawing. In: Proceedings of the 12th Annual ACM-SIAM symposium on Discrete algorithms, pp. 506–515 (2001)

[8] Chuang, R.C.-N., Garg, A., He, X., Kao, M.-Y., Lu, H.-I.: Compact encodings of planar graphs via canonical orderings and multiple parentheses. In: Proceedings of the 25th International Colloquium on Automata, Languages and Programming, pp. 118–129 (1998)

[9] Chung, F.R.K., Leighton, F.T., Rosenberg, A.L.: Embedding graphs in books: a layout problem with applications to VLSI design. SIAM J. Algebr. Discrete Methods 8(1), 33–58 (1987)

[10] Gavoille, C., Hanusse, N.: On compact encoding of pagenumber k graphs. Discrete Mathematics & Theoretical Computer Science (to appear, 2007)

[11] Isenburg, M., Snoeyink, J.: Face fixer: Compressing polygon meshes with properties. In: Proceedings of SIGGRAPH 2000, pp. 263–270 (2000)

[12] Jacobson, G.: Space-efficient static trees and graphs. In: Proceedings of the 30th Annual Symposium on Foundations of Computer Science, pp. 549–554 (1989)

[13] Lu, H.-I., Yeh, C.-C.: Balanced parentheses strike back. ACM Transactions on Algorithms (accepted, 2007)

[14] Munro, J.I., Raman, V.: Succinct representation of balanced parentheses and static trees. SIAM J. Comput. 31(3), 762–776 (2001)
[15] Rosenberg, A.L.: The diogenes design methodology: toward automatic physical layout. In: Proceedings of the International Workshop on Parallel Algorithms & Architectures, pp. 335–348. North-Holland Publishing Co., Amsterdam (1986)
[16] Schnyder, W.: Embedding planar graphs on the grid. In: Proceedings of the 1st Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 138–148 (1990)
[17] Tarjan, R.E.: Sorting using networks of queues and stacks. J. Assoc. Comput. Mach. 19, 341–346 (1972)
[18] Yannakakis, M.: Four pages are necessary and sufficient for planar graphs. In: Proceedings of the 18th Annual ACM-SIAM Symposium on Theory of Computing, pp. 104–108 (1986)

More Efficient Algorithms and Analyses for Unequal Letter Cost Prefix-Free Coding

Mordecai Golin* and Jian Li**

[1] Dept of Computer Science & Engineering, HKUST, Hong Kong, China
golin@cs.ust.hk.
[2] Dept of Computer Science & Engineering, Fudan University, Shanghai, China
lijian83@fudan.edu.cn.

Abstract. There is a large literature devoted to the problem of finding an optimal (min-cost) prefix-free code with an unequal letter-cost encoding alphabet of size. While there is no known polynomial time algorithm for optimally solving it, there are many good heuristics that all provide additive errors to optimal. The additive error in these algorithms usually depends linearly upon the size of the largest encoding letter.

This paper was motivated by the problem of finding optimal codes when the encoding alphabet is infinite. Because the largest letter cost is infinite, the previous analyses could give infinite error bounds. We provide a new algorithm that works with infinite encoding alphabets. When restricted to the finite alphabet case, our algorithm often provides better error bounds than the best previous ones known.

1 Introduction

Let $\Sigma = \{\sigma_1, \sigma_2 \ldots, \sigma_t\}$ be an *encoding alphabet*. Word $w \in \Sigma^*$ is a *prefix* of word $w' \in \Sigma^*$ if $w' = wu$ where $u \in \Sigma^*$ is a non-empty word. A *Code* over Σ is a collection of words $C = \{w_1, \ldots, w_n\}$. Code C is *prefix-free* if for all $i \neq j$ w_i is not a prefix of w_j. See Figure 1.

Let $cost(w)$ be the *length* or number of characters in w. Given a set of associated probabilities $p_1, p_2, \ldots, p_n \geq 0$, $\sum_i p_i = 1$, the cost of the code is $Cost(C) = \sum_{i=1}^{n} cost(w_i)p_i$. The *prefix coding* problem, sometimes known as the *Huffman encoding* problem is to find a prefix-free code over Σ of minimum cost. This problem is very well studied and has a well-known $O(tn \log n)$-time greedy-algorithm due to Huffman [13] ($O(tn)$-time if the p_i are sorted in non-decreasing order).

One well studied generalization of the problem is to let the encoding letters have different costs. That is, let $\sigma_i \in \Sigma$ have associated cost c_i. The cost of codeword $w = \sigma_{i_1}\sigma_{i_2}\ldots\sigma_{i_l}$ will be $cost(w) = \sum_{k=1}^{l} c_{i_k}$, i.e., the sum of the costs of its letters (rather than the length of the codeword) with the cost of the code still being defined as $Cost(C) = \sum_{i=1}^{n} cost(w_i)p_i$ with this new cost function.

* Work partially supported by HK RGC Competitive Research Grant 613105.
** Work done while visiting HKUST.

T. Tokuyama (Ed.): ISAAC 2007, LNCS 4835, pp. 329–340, 2007.

x	aaa	aab	ab	b
$cost(x)$	3	5	4	3

x	aaa	aab	ab	$aaba$
$cost(x)$	3	5	4	6

Fig. 1. Two codes for $\Sigma = \{a, b\}$. Code $\{aaa, aab, ab, b\}$ is prefix-free. Code $\{aaa, aab, ab, aaba\}$ is not prefix-free because aab is a prefix of $aaba$. The second row of the tables contain the costs of the codewords when $cost(a) = 1$ and $cost(b) = 3$.

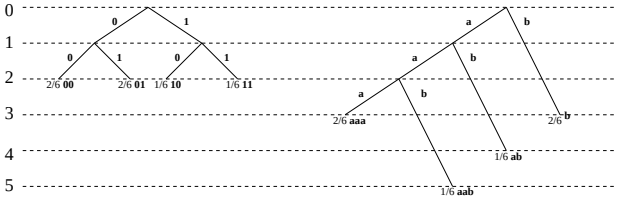


Fig. 2. Two min-cost prefix-free codes for probabilities $2/6, 2/6, 1/6, 1/6$ and their tree representations. The code on the left is optimal for $c_1 = c_2 = 1$ while the code on the right, the prefix-free code from Figure 1, is optimal for $c_1 = 1, c_2 = 3$.

The existing, large, literature on the problem of finding a minimal-cost prefix-free code when the c_i are no longer equal, which will be surveyed below, assumes that Σ is a finite alphabet, i.e., that $t = |\Sigma| < \infty$. The original motivation of this paper was to address the problem when Σ is *unbounded*, which, as will briefly be described in Section 3 models certain types of language restrictions on prefix-free codes and the imposition of different cost metrics on search trees. The tools developed, though, turn out to provide improved approximation bounds for many of the finite cases as well. More specifically, it was known [16,18][1] that $\frac{1}{c}H(p_1, \ldots, p_n) \leq OPT$ where $H(p_1, \ldots, p_n) = -\sum_{i=1}^{n} p_i \log p_i$ is the *entropy* of the distribution, c is the unique positive root of the *characteristic equation* $1 = \sum_{i=1}^{t} 2^{-cc_i}$ and OPT is the minimum cost of any prefix-free code for those p_i. Note that in this paper, $\log x$ will always denote $\log_2 x$.

The known efficient algorithms create a code T that satisfies

$$C(T) \leq \frac{1}{c}H(p_1, \ldots, p_n) + f(\mathcal{C}) \tag{1}$$

where $C(T)$ is the cost of code T, $\mathcal{C} = (c_1, c_2, \cdots, c_t)$ and $f(\mathcal{C})$ is some function of the letter costs $\mathcal{C}$, with the actual value of $f(\mathcal{C})$ depending upon the particular algorithm. Since $\frac{1}{c}H(p_1, \ldots, p_n) \leq OPT$, code T has an *additive error* at most $f(\mathcal{C})$ from OPT. The $f(\mathcal{C})$ corresponding to the different algorithms shared an almost linear dependence upon the value $c_t = \max(\mathcal{C})$, the largest letter cost. They therefore can not be used for infinite $\mathcal{C}$. In this paper we present a new

[1] Note that if $t = 2$ with $c_1 = c_2 = 1$ then $c = 1$ and this reduces to the standard entropy lower bound for prefix-free coding. Although the general lower bound is usually only explicitly derived for finite t, Krause [16] showed how to extend it to infinite t in cases where a positive root of $1 = \sum_{i=1}^{\infty} 2^{-cc_i}$ exists.

algorithmic variation (all algorithms for this problem start with the same splitting procedure so they are all, in some sense, variations of each other) with a new analysis:

- (Theorems 2 and 3) For finite $\mathcal{C}$ we derive new additive error bounds $f(\mathcal{C})$ which in many cases, are much better than the old ones.
- (Lemma 8) If $\mathcal{C}$ is infinite but $d_j = |\{m \mid j \leq c_m < j+1\}|$ is bounded, then we can still give a bound of type (1).
- (Theorem 4) If $\mathcal{C}$ is infinite but d_i is unbounded then we can not provide a bound of type (1) but, as long as $\sum_{i=1}^{\infty} c_m 2^{-cc_m} < \infty$, we can show that

$$\forall \epsilon > 0, \quad C(T) \leq (1+\epsilon)\frac{1}{c}H(p_1, \ldots, p_n) + f(\mathcal{C}, \epsilon) \tag{2}$$

  where $f(\mathcal{C}, \epsilon)$ is some constant based only on $\mathcal{C}$ and ϵ.

We now provide some more history and motivation. The unequal letter cost coding problem was originally motivated by coding problems in which different characters have different transmission times or storage costs see e.g., [15]. Blachman [4], Marcus [17], and (much later) Gilbert [9] give heuristic constructions without analyses of the costs of the codes they produced. Karp gave the first algorithm yielding an exact solution (assuming the letter costs are integers); Karp's algorithm transforms the problem into an integer program and does not run in polynomial time [15]. Later exact algorithms based on dynamic programming were given by Golin and Rote [10] for arbitrary t and a slightly more efficient one by Bradford et. al. [5] for $t = 2$.. These algorithms run in $n^{\theta(c_t)}$ time where c_t is the cost of the largest letter. Despite the extensive literature, there is no known polynomial-time algorithm for the generalized problem, nor is the problem known to be NP-hard. Golin, Kenyon and Young [12] provide a polynomial time approximation scheme (PTAS). Their algorithm is mainly theoretical and not useful in practice. Finally, in contrast to the non-alphabetic case, alphabetic coding has a polynomial-time algorithm $O(tn^3)$ time algorithm [14].

Karp's result was followed by many efficient algorithms [16,8,7,18,2]. As mentioned above, $\frac{1}{c}H(p_1, \ldots, p_n) \leq OPT$; almost all of these algorithms produce codes of cost at most $C(T) \leq \frac{1}{c}H(p_1, \ldots, p_n) + f(\mathcal{C})$ and therefore give solutions within an *additive error* of optimal. An important observation is that the additive error in these papers $f(\mathcal{C})$ somehow incorporate the cost of the largest letter $c_t = \max(\mathcal{C})$. Typical in this regard is Mehlhorn's algorithm [18] which provides a bound of

$$cC(T) - H(p_1, \ldots, p_n) \leq (1 - p_1 - p_n) + cc_t \tag{3}$$

Thus, none of the algorithms described can be used to address infinite alphabets with unbounded letter costs.

In this paper we are only interested in the general coding problem and not the *alphabetic* one[2] and will therefore have freedom to dictate the original order

[2] *Alphabetic coding* is the same problem with the additional constraint that the codewords must be chosen in increasing alphabetic order (with respect to the words to be encoded).

in which the p_i are given and the ordering of the c_m. We will actually always assume that $p_1 \geq p_2 \geq p_2 \geq \cdots$ and $c_1 \leq c_2 \leq c_3 \leq \cdots$. These assumptions are the starting point that will permit us to derive better bounds. Furthermore, for simplicity, we will always assume that $c_1 = 1$. If not, we can always force this by uniformly scaling all of the c_i.

For further references on Huffman coding with unequal letter costs, see Abrahams' survey on source coding [1, Section 2.7], which contains a section on the problem.

Due to lack of space in this extended abstract, most of the proofs have been omitted. They are available in the full paper [11].

2 Notations and Definitions

There is a very standard correspondence between prefix-free codes over alphabet Σ and $|\Sigma|$-ary trees in which the m^{th} child of node v is labelled with character $\sigma_m \in \Sigma$. A path from the root in a tree to a leaf will correspond to the word constructed by reading the edge labels while walking the path. Because of this correspondence we will speak about codes and trees interchangeably.

Definition 1. *Let C be a prefix-free code over Σ and T its associated tree. N_T will denote the set of* internal nodes *of T.*

Definition 2. *Set c to be the unique positive solution to $1 = \sum_{i=1}^{t} 2^{-cc_i}$. Note that if $t < \infty$, then c must exists while if $t = \infty$, c might not exist. We only define c for the cases in which it exists. c is sometimes called the* root of the characteristic equation *of the letter costs.*

Definition 3. *Given letter costs c_i and their associated characteristic root c, let T be a code with those letter costs. If $p_1, p_2, \ldots, p_n \geq 0$ is a probability distribution then the* redundancy *of T relative to the p_i is $R(T; p_1, \ldots, p_n) = C(T) - \frac{1}{c}H(p_1, \ldots, p_n)$. We will also define the* normalized redundancy *to be $\mathrm{NR}(T; p_1, \ldots, p_n) = cR = cC(T) - H(p_1, \ldots, p_n)$. If the p_i and T are understood, we will write $R(T)$ ($\mathrm{NR}(T)$) or even R (NR).*

3 Examples of Unequal-Cost Letters

It is not a-priori as clear why infinite alphabets would be interesting. We now discuss some motivation.

In what follows we will need some basic language notation. A language $\mathcal{L}$ is just a set of words over alphabet Σ. The *concatenation* of languages A and B is $AB = \{ab \mid a \in A, b \in B\}$. The i-fold concatenation, $\mathcal{L}^i$, is defined by $\mathcal{L}^0 = \{\lambda\}$ (the language containing just the empty string), $\mathcal{L}^1 = \mathcal{L}$ and $\mathcal{L}^i = \mathcal{L}\mathcal{L}^{i-1}$. The *Kleene star* of $\mathcal{L}$, is $\mathcal{L}^* = \bigcup_{i=0}^{\infty} \mathcal{L}^i$.

We start with cost vector $\mathcal{C} = \{1, 2, 3, \ldots, \}$ i.e, $\forall m > 0, c_m = m$. An early use of this problem was [19]. The idea there was to construct a tree (not a code)

in which the internal pointers to children were stored in a linked list. Taking the m^{th} pointer corresponds to using character σ_m. The time that it takes to *find* the m^{th} pointer is proportional to the location of the pointer in the list. Thus (after normalizing time units) $c_m = m$.

We now consider a generalization of the problem of **1**-ended codes. The problem of finding min-cost prefix-free code with the additional restriction that all codewords end with a 1 was studied in [3,6] with the motivation of designing self-synchronizing codes. One can model this problem as follows. Let $\mathcal{L}$ be a language. In our problem, $\mathcal{L} = \{w \in \{0,1\}^* \mid \text{the last letter in } w \text{ is a } \mathbf{1}\}$. We say that a code C is in $\mathcal{L}$ if $C \subseteq \mathcal{L}$. The problem is to find a minimum cost code among all codes in $\mathcal{L}$.

Now suppose further that $\mathcal{L}$ has the special property that $\mathcal{L} = \mathcal{Q}^*$ where $\mathcal{Q}$ is itself a prefix-free language. Then every word in $\mathcal{L}$ can be uniquely decomposed as the concatenation of words in $\mathcal{Q}$. If the decomposition of $w \in \mathcal{L}$ is $w = q_1 q_2 \dots q_r$ for $q_i \in \mathcal{Q}$ then $cost(w) = \sum_{i=1}^{r} cost(q_i)$. We can therefore model the problem of finding a minimum cost code among all codes in $\mathcal{L}$ by first creating an infinite alphabet $\Sigma_{\mathcal{Q}} = \{\sigma_q \mid q \in Q\}$ with associated cost vector $C_{\mathcal{Q}}$ (in which the length of σ_q is $cost(q)$) and then solving the minimal cost coding problem for $\Sigma_{\mathcal{Q}}$ with those associated costs. For the example of **1**-ended codes we set $\mathcal{Q} = \{1, 01, 001, 0001, \dots\}$ and thus have $\mathcal{C} = \{1, 2, 3, \dots, \}$ i.e, an infinite alphabet with $c_m = m$ for all $m \geq 1$.

Now consider generalizing the problem as follows. Suppose we are given an unequal cost coding problem with *finite* alphabet $\Sigma = \{\sigma_1, \dots, \sigma_t\}$ and associated cost vector $\mathcal{C} = (c_1, \dots, c_t)$. Now let $\Sigma' \subset \Sigma$ and define $\mathcal{L} = \Sigma^*\Sigma' = \{w \in \Sigma^* \mid$ the last letter in w is in $\Sigma'\}$. Now note that $\mathcal{L} = D^*$ where $D = (\Sigma - \Sigma')^*\Sigma'$ is a prefix-free language. We can therefore model the problem of finding a minimum cost code among all codes in $\mathcal{L}$ by solving an unequal cost coding problem with alphabet Σ_D and $\mathcal{C}_D$. The important observation is that $d_j = |\{d \in \Sigma_D \mid cost(d) = j\}|$, the number of letters in Σ_D of length j, satisfies a linear recurrence relation. Bounding redundancies for these types of $\mathcal{C}$ will be discussed in Section 6, Case 4.

As an illustration, consider $\Sigma = \{1, 2, 3\}$ with $\mathcal{C} = (1, 1, 2)$ and $\Sigma' = \{1\}$; our problem is find minimal cost prefix-free codes in which all words end with a 1. $\mathcal{L} = \{1, 2, 3\}^*\{1\} = D^*$, where $D = \{2, 3\}^*\{1\}$. The number of characters in Σ_D with length j is $d_1 = 1$, $d_2 = 1$, $d_3 = 2$, $d_4 = 3$, $d_5 = 5$, and, in general, $d_{i+2} = d_{i+1} + d_i$, so $d_i = F_i$, the Fibonacci numbers.

4 The Algorithm

All of the provably efficient heuristics for the problem, e.g., [16,8,7,18,2], use the same basic approach, which itself is a generalization of Shannon's original binary splitting algorithm [20]. The idea is to create t *bins*, where bin m has weight 2^{-cc_m} (so the sum of all bin weights is 1). The algorithms then try to partition the probabilities into the bins; bin m will contain a set of contiguous probabilities $p_{l_m}, p_{l_m+1}, \dots, p_{r_m}$ whose sum will have total weight "close" to

2^{-cc_m}. The algorithms fix the first letter of all the codewords associated with the p_k in bin m to be σ_m. After fixing the first letter, the algorithms then recurse, normalizing $p_{l_m}, p_{l_m+1}, \ldots, p_{r_m}$ to sum to 1, taking them as input and starting anew. The various algorithms differ in how they group the probabilities and how they recurse.

Here we use a generalization of the version introduced in [18]. The algorithm first preprocesses the input and calculates all $P_k = p_1 + p_2 + \ldots + p_k$ ($P_0 = 0$)and $s_k = p_1 + p_2 + \ldots + p_{k-1} + \frac{p_k}{2}$. Note that if we lay out the p_i along the unit interval in order, then s_k can be seen as the *midpoint* of interval p_i. It then partitions the probabilities into ranges, and for each range it constructs left and right boundaries L_m, R_m. p_k will be assigned to bin m if it "falls" into the "range" $[L_m, R_m)$.

If the interval p_k falls into the range, i.e., $L_m \leq P_{k-1} < P_k < R_m$ then p_k should definitely be in bin m. But what if p_k spans two (or more) ranges, e.g., $L_m \leq P_{k-1} < R_m < P_k$? To which bin should p_k be assigned? The choice made by [18] is that p_k is assigned to bin m if $s_k = p_1 + p_2 + \ldots + p_k/2$ falls into $[L_m, R_m)$, i.e., the midpoint of p_k falls into the range.

Our procedure $CODE(l, r, U)$ will build a prefix-free code for $p_l, \ldots, p_r$ in which every code word starts with prefix U. To build the entire code we call $CODE(1, n, \lambda)$, where λ is the empty string. Figure 3 gives pseudocode.

$CODE(l, r, U)$;
{*Constructs codewords* $U_l, U_{l+1}, \ldots, U_r$ *for* $p_l, p_{l+1}, \ldots, p_r$.
U *is previously constructed common prefix of* $U_l, U_{l+1}, \ldots, U_r$.}
If $l = r$
    then codeword U_l is set to be U.
else {*Distribute* p_i*s into initial bins* I_m^*}
    $L = P_{l-1}$; $R = P_r$; $w = R - L$
    $\forall m$, let $L_m = L + w\sum_{i=1}^{m-1} 2^{-cc_i}$ and $R_m = L_m + w2^{-cc_m}$.
        set $I_m^* = \{k \mid L_m \leq s_k < R_m\}$ }
    {*Shift the bins to become final* I_m. *Afterwards,*
    all bins $> M$ *are empty, all bins* $\leq M$ *non-empty and* $\forall m \leq M$, $I_m = \{l_m, \ldots r_m\}$}
    {*shift left so there are no empty "middle" bins.*}
    $M = 0$; $k = l$;
    while $k \leq r$ **do**
        $M = M + 1$;
        $l_M = k$; $r_M = \max\left(\{k\} \bigcup \{i > k \mid i \in I_M^*\}\right)$;
        $k = r_M + 1$;
    {*If all* p_i*'s are in first bin, shift* p_r *to* 2^{nd} *bin* }
    if $r_1 = r$ **then**
        $M = 2$; $r_1 = r - 1$; $l_2 = r_2 = r$;
    for $m = 1$ **to** M **do**
        $CODE(l_m, r_m, U\sigma_m)$;

Fig. 3. Our algorithm. Note that the first step of creating the I_m^* was written to simplify the development of the analysis. In practice, it is not actually constructed.

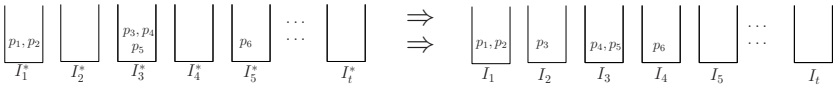


Fig. 4. The splitting procedure creates the bins I_m^* on the left. The shifting procedure then creates the I_m on the right.

Assume that we currently have a prefix of U assigned to $p_l, \ldots, p_r$. Let v be node in the tree associated with U. Let $w(v) = \sum_{k=l}^{r} p_k$.

(i) If $l = r$ then word U is assigned to p_l. Correspondingly, v is a leaf in the tree with weight $w(v) = p_l$.

(ii) Otherwise let $L = P_{l-1}$ and $R = P_r$. Split $R - L = w(v)$ into t ranges[3] as follows. $\forall 1 \leq m \leq t, \quad L_m = L + (R - L) \sum_{i=1}^{m-1} 2^{-cc_i}, \quad R_m = L + (R - L) \sum_{i=1}^{m} 2^{-cc_i}$. Insert p_k, $l \leq k \leq r$ in bin m if $s_k \in [L_m, R_m)$. Bin m will thus contain the p_k in $I_m^*(v) = \{k \mid L_m \leq s_k < R_m\}$.

We now shift the items p_k *leftward* in the bins as follows. Walk through the bins from left to right. If the current bin already contains some p_k, continue to the next bin. If the current bin is empty, take the first p_k that appears in a bin to the right of the current one, shift p_k into the current bin and walk to the next bin. Stop when all p_k have been seen. Let $I_m(v)$ denote the items in the bins after this shifting. See figure 4.

We then check if all of the items are in $I_1(v)$. If they are, we take p_r and move it into $I_2(v)$ (and set $M(v) = 2$).

Finally, after creating the all of the $I_m(v)$ we let $l_m = \min\{k \in I_m(v)\}$ and $r_m = \max\{k \in I_m(v)\}$ and recurse, for each $m < M(v)$ building $CODE(l_m, r_m, U\sigma_m)$

Indeed, we can show the algorithm can be implemented with a running time bounded by $n + \sum_{v \in N_T} \log nM(v) = O(n \log n)$ with no dependence upon t. The high level idea is we do not actually construct the bins $I_m^*(v)$ first. We can more efficiently construct the $I_m(v)$ using a binary search each time. The implementation details are omitted here and can be found in the full paper [11].

For comparison, we point out the algorithm in [18] also starts by first finding the $I_m^*(v)$. Since it assumed $t < \infty$, its shifting stage was much simpler, though. It just shifted p_l into the first bin and p_r into the t^{th} bin (if they were not already there).

We will now see that our modified shifting procedure not only permits a finite algorithm for infinite encoding alphabets, but, in conjunction with the added assumption that the p_i are sorted in non-decreasing order, also often provides a provably better approximation for *finite* encoding alphabets.

5 Analysis

In the analysis we define $w_m^*(v) = \sum_{k \in I_m^*(v)} p_k$, $w_m(v) = \sum_{k \in I_m(v)} p_k$. Note that $w(v) = \sum_{m=1}^{t} w_m^*(v) = \sum_{m=1}^{t} w_m(v) = \sum_{k=l}^{r} p_k$. We first need some Lemmas from [18].

[3] In the description, t is permitted to be finite or infinite.

Lemma 1. *[18] Let T be a code tree and N_T be the set of all* internal *nodes of T. Then*

1. *The cost $C(T)$ of the code tree T is $C(T) = \sum_{v\in N_T}\sum_{m=1}^{t} c_m \cdot w_m(v)$.*
2. *The entropy $H(p_1, p_2, \ldots, p_n)$ is $H(p_1, p_2, \ldots, p_n) = \sum_{v\in N_T} w(v) \cdot H\left(\frac{w_1(v)}{w(v)}, \frac{w_2(v)}{w(v)}, \ldots\right)$.*

Lemma 1 permits expressing the normalized redundancy of T as

$$NR(T)=c\cdot C(T)-H(p_1,p_2,\ldots,p_n)=\sum_{v\in N_T} w(v)\left[\sum_{m=1}^{t}\frac{w_m(v)}{w(v)}\left(\log 2^{cc_m}+\log\frac{w_m(v)}{w(v)}\right)\right].$$

Set $E(v,m) = \frac{w_m(v)}{w(v)}\left(\log 2^{cc_m} + \log\frac{w_m(v)}{w(v)}\right)$.

Note that $\mathrm{NR}(T) = \sum_{v\in N_T} w(v)\left(\sum_{m=1}^{t} E(v,m)\right)$. For convenience we will also define $E^*(v,m) = \frac{w^*_m(v)}{w(v)}\left(\log 2^{cc_m} + \log\frac{w^*_m(v)}{w(v)}\right)$ and $\mathrm{NR}^*(T) = \sum_{v\in N_T} w(v)\left(\sum_{m=1}^{t} E^*(v,m)\right)$. The analysis proceeds by bounding the values of $\mathrm{NR}^*(T)$ and $\mathrm{NR}(T) - \mathrm{NR}^*(T)$.

Lemma 2. *[18][4] (*note: In this Lemma, the p_i can be arbitrarily ordered.*) Consider any call $CODE(l, r, U)$ with $l < r$. Let node v correspond to the word U. Let sets $I^*_1, I^*_2, \ldots$ be defined as in procedure CODE.*

a) *If $I^*_m = \emptyset$, then $w^*_m(v) = 0$.*
b) *If $I^*_m = \{e\}$. then $w^*_m(v) = p_e$.*
c) *If $|I^*_m| \geq 2$. Let $e = \min I^*_m$ and $f = \max I^*_m$. $E^*(v,m) \leq \frac{p_e+p_f}{w(v)}$. Furthermore, if $m = 1$ then $E^*(v,m) \leq \frac{p_f}{w(v)}$, while if $m = t$, then $E^*(v,m) \leq \frac{p_e}{w(v)}$.*

Corollary 3. *If the p_i are sorted in nonincreasing order then in case (c) of Lemma 2, if $m = 1$, $E^*(v,m) \leq \frac{p_f}{w(v)}$, while if $m > 1$, then $E^*(v,m) \leq \frac{2p_e}{w(v)}$.*

The following lemma bounds the gap between NR and NR^*.

Lemma 4. $\mathrm{NR} - \mathrm{NR}^* \leq c(c_2 - c_1)\sum_{i\in A} p_i$ *where $A = \{i \mid i$ is right shifted at some step$\}$.*

Lemma 5. $\mathrm{NR}^* \leq 2(1 - p_1) + \sum_{v\in N_T}\sum_{\substack{1\leq m\leq t\\ |I^*_m(v)|=1}} w(v)E^*(v,m)$.

Combining this Lemma with Lemma 4 gives

Corollary 6. $\mathrm{NR}\leq 2(1-p_1)+c(c_2-c_1)\sum_{i\in A} p_i+\sum_{v\in N_T}\sum_{\substack{1\leq m\leq t\\ |I^*_m(v)|=1}} w(v)E^*(v,m)$.

We will now see different bounds on the last summand in the above expression. Section 6 compares the results we get to previous ones for different classes of $\mathcal{C}$. Before proceeding, we note that any p_i can only appear as $I^*_m(v) = \{p_i\}$ for at

[4] Slightly rewritten for our notation.

most one (m, v) pair. Furthermore, if p_i does appear in such a way, then it can not have been made a leaf by a previous right shift and thus $p_i \notin A$.

We start by noting that, when $t \leq \infty$ our bound is never worse than 1 plus the old bound of $(1 - p_1 - p_n) + cc_t$ stated in (3).

Theorem 1. *If $t < \infty$ then* $\mathrm{NR} \leq 2(1 - p_1) + cc_t$.

For a tighter analysis we will need a better bound for the case $|I^*_m| = 1$. The following simple lemma is a direct result from our splitting procedure.

Lemma 7. *(a) Let $v \in N_T$. Suppose i is such that $i \in I^*_m(v)$. then $\frac{p_i}{w(v)} \leq 2 \cdot \sum_{j=m}^{t} \frac{1}{2^{c \cdot c_j}}$. (b) Further suppose there is some $m' > m$ such that $I^*_{m'} \neq \emptyset$. Then $\frac{p_i}{w(v)} \leq 2 \cdot \sum_{j=m}^{m'} \frac{1}{2^{c \cdot c_j}} \leq 4 \cdot \sum_{j=m}^{m'-1} \frac{1}{2^{c \cdot c_j}}$.*

Definition 4. *Set $\beta_m = 2^{cc_m} \sum_{i=m}^{t} 2^{-cc_i}$ and $\beta = \sup\{\beta_m \mid 1 \leq m \leq t\}$*

We can now show our first improved bound.

Theorem 2. *If $\beta < \infty$ then* $\mathrm{NR} \leq 2(1 - p_1) + \max\big(c(c_2 - c_1), 1 + \log \beta\big)$.

This immediately gives an improved bound for many finite cases because, if $t < \infty$, then $\beta_m = 2^{cc_m} \sum_{i=m}^{t} 2^{-cc_i} \leq t - m + 1$ so $\beta \leq t$. Thus

Theorem 3. *If t is finite then* $\mathrm{NR} \leq 2(1 - p_1) + \max\big(c(c_2 - c_1), 1 + \log t\big)$.

Definition 5. *For all $j \geq 1$, set $d_j = |\{i \mid j \leq c_i < j + 1\}|$.*

This permits us to give another general bound that also works for many infinite alphabets. By evaluating β and from Theorem 2, we can get the following simple lemma.

Lemma 8. *If $d_j = O(1)$, then $NR = O(1)$. In particular, if $\forall j$, $d_j \leq K$ then $\beta \leq \frac{2^c K}{1 - 2^{-c}}$ so, from Theorem 2, $NR \leq 2(1-p_1) + \max\left(c(c_2 - c_1),\ 1 + c + \log\left(\frac{K}{1-2^{-c}}\right)\right)$.*

For general infinite alphabets we are not able to derive a constant redundancy bound but we obtain the following theorem.

Theorem 4. *If $\mathcal{C}$ is infinite and $\sum_{m=1}^{\infty} c_m 2^{-cc_m} < \infty$, then, for every $\epsilon > 0$*

$$R \leq \epsilon \frac{1}{c} H(p_1, \ldots, p_n) + f(\mathcal{C}, \epsilon) \tag{4}$$

where $f(\mathcal{C}, \epsilon)$ is some constant based only on $\mathcal{C}$ and ϵ. Note that this is equivalent to stating that $C(T) \leq (1 + \epsilon)OPT + f(\mathcal{C}, \epsilon)$.

6 Examples

We now examine some of the bounds derived in the last section and show how they compare to the old bound of $(1 - p_1 - p_n) + cc_t$ stated in (3). In particular,

we show that for large families of costs the old bounds go to infinity while the new ones give uniformly constant bounds.

Case 1: $\mathcal{C}_\alpha = (c_1, c_2, \ldots, c_{t-1}, \alpha)$ with $\alpha \uparrow \infty$:
We assume $t \geq 3$ and all of the c_i, $i < t$, are fixed. Let $c^{(\alpha)}$ be the root of the corresponding characteristic equation $1 = 2^{-c\alpha} + \sum_{i=1}^{t-1} c^{-cc_i}$. Note that $c^{(\alpha)} \downarrow \bar{c}$ where $\bar{c}$ is the root of $1 = \sum_{i=1}^{t-1} c^{-cc_i}$. Let (NR_α) R_α be the (normalized) redundancy corresponding to $\mathcal{C}_\alpha$.

For any fixed α, the old bound (3) would give that both of NR_α and R_α tend to ∞ as α increases. Compare this to Theorem 3 which gives a uniform bound of $NR_\alpha \leq 2(1-p_1) + \max\left(c^{(c_{t-1})}(c_2 - c_1), 1 + \log t\right)$ and $R_\alpha \leq \frac{\mathrm{NR}_\alpha}{c^{(\alpha)}} \leq \frac{2(1-p_1)+\max\left(c^{(c_{t-1})}(c_2-c_1),1+\log t\right)}{\bar{c}}$.

Example 1. *Let $t = 3$ with $c_1 = c_2 = 1$ and $c_3 = \alpha \geq 1$. The old bounds (3) gives an asymptotically infinite error as $\alpha \to \infty$. The bound from Theorem 3 is $NR_\alpha \leq 2(1-p_1) + \max\left(c^{(\alpha)}(c_2 - c_1),\, 1 + \log t\right) \leq 3 + \log 3$ independent of α. Since $c^{(\alpha)} \geq \bar{c} = 1$ we also get $R_\alpha = \frac{NR_\alpha}{c^{(\alpha)}} \leq 3 + \log 3$.*

Case 2: A finite alphabet that approaches an infinite one.
Let $\mathcal{C}$ be an infinite sequence of letter costs such that there exists a $K > 0$ satisfying for all j, $d_j = |\{i \mid j \leq c_i < j\}| \leq K$. Let c be the root of the characteristic equation $1 = \sum_{i=1}^{\infty} 2^{-cc_i}$. Let $\Sigma^{(t)} = \{\sigma_1, \ldots, \sigma_t\}$ and its associated letter costs be $\mathcal{C}^{(t)} = \{c_1, \ldots, c_t\}$. Let $c^{(t)}$ be the root of the corresponding characteristic equation $1 = \sum_{i=1}^{t} 2^{-cc_i}$ and (NR_t) R_t be the associated (normalized) redundancy. Note that $c^{(t)} \uparrow c$ as t increases.

For any fixed t, the old bound (3) would be $NR_t \leq (1 - p_1 - p_n) + c^{(t)} c_t$ which goes to ∞ as t increases. Lemma 8 tells us that $\beta^{(t)} = \max_{1 \leq m \leq t} 2^{c^{(t)} c_m} \sum_{i=1}^{t} 2^{c^{(t)} c_i} \leq \frac{2^c K}{1 - 2^{-c^{(t)}}} \leq \frac{2^c K}{1 - 2^{-c^{(2)}}}$. so, from Theorem 2 and the fact that $\forall t$, $c^{(2)} \leq c^{(t)} < c$, we get $NR_t \leq 2(1-p_1) + \max\left(c(c_2 - c_1),\, 1 + c + \log \frac{K}{1 - 2^{-c^{(2)}}}\right)$.

Example 2. *Let $\mathcal{C} = (1, 2, 3, \ldots)$. i.e., $c_m = m$. The old bounds (3) gives an asymptotically infinite error as $\alpha \to \infty$. For this case $c = 1$ and $K = 1$. $c^{(2)}$ is the root of the characteristic equation $1 = 2^{-2} + 2^{-2c}$. Solving gives $2^{-c^{(2)}} = \frac{\sqrt{5}-1}{2}$ and $c^{(2)} = 1 - \log(\sqrt{5} - 1) \approx 0.694\ldots$. Plugging into our equations gives $NR_t \leq 4.388$ and $R_t \leq 6.232$.*

Case 3: An infinite case when $d_j = O(1)$. This permits applying Lemma 8 directly.

Example 3. *Let $\mathcal{C}$ contain d copies each of $i = 1, 2, 3, \ldots$, i.e., $c_m = 1 + \lfloor \frac{m-1}{d} \rfloor$. Note that $K = d$. If $d = 1$, i.e., $c_m = m$, then $c = K = 1$ and $R = \mathrm{NR} \leq 2(1-p_1) + 2$. If $d > 1$ then $A(x) = \sum_{m=1}^{\infty} c_m z^m = \frac{dz}{1-z}$. The solution α to $A(\alpha) = 1$ is $\alpha = \frac{1}{d+1}$, so $c = -\log \alpha = \log(d+1)$. The lemma gives $\mathrm{NR} \leq 2(1-p_1) + \left(1 + \log\left(\frac{K}{1-2^{-c}}\right)\right) \leq 3 + \log(d+1)$, $\quad R \leq 1 + \frac{3}{\log(d+1)}$.*

Case 4: d_j are integral and satisfy a linear recurrence relation.
In this case the generating function $A(z) = \sum_{j=1}^{\infty} d_j z^j = \sum_{m=1}^{\infty} z^{c_m}$ can be written as $A(z) = \frac{P(z)}{Q(z)}$ where $P(z)$ and $Q(z)$ are relatively prime polynomials. Let γ be a smallest modulus root of $Q(z)$. If γ is the unique root of that modulus (which happens in most interesting cases) then it is known that $d_j = \Theta(j^{d-1}\gamma^{-j})$ (which will also imply that γ is positive real) where d is the multiplicity of the root. There must then exist some $\alpha < \gamma$ such that $A(\alpha) = 1$. By definition $c = -\log \alpha$. Furthermore, since $\alpha < \gamma$ we must have that $\sum_{j=1}^{\infty} d_j j \alpha^j = \sum_{m=1}^{\infty} c_m \alpha^{c_m}$ also converges, so Theorem 4 applies.

Note that $h(x) = \sum_{j=x}^{\infty} d_j j \alpha^j = O\left(\sum_{j=x}^{\infty} j^{d-1} j \left(\frac{\alpha}{\gamma}\right)^j\right) = O\left(x^d \left(\frac{\alpha}{\gamma}\right)^x\right)$, implying $h^{-1}(\epsilon) = \log_{\gamma/\alpha} 1/\epsilon + O(\log\log 1/\epsilon)$ where we define $h^{-1}(\epsilon) = \max\{x \mid h(x) \leq \epsilon, h(x-1) > \epsilon\}$. Working through the proof of Theorem 4 we find that when the c_m are all integral, for all m', $g(m') = \sum_{m \geq m'} c_m \alpha^{c_m} \leq \sum_{j \geq c_{m'}} j d_j \alpha_j = h(c_{m'})$. Recall that $m_\epsilon = \max\{m \mid c_m \leq N_\epsilon\}$. Then $g(m_\epsilon) \leq h(N_\epsilon)$. Since $g(m_\epsilon) \leq \epsilon/6$, $N_\epsilon \leq h^{-1}(\epsilon/6) = \log_{\gamma/\alpha} 1/\epsilon + O(\log\log 1/\epsilon)$ and thus our algorithm creates a code T satisfying

$$C(T) - OPT \leq \epsilon OPT + \log_{\gamma/\alpha} 1/\epsilon + O(\log\log 1/\epsilon). \tag{5}$$

Example 4. *Consider the case where $d_j = F_j$, the j^{th} Fibonacci number, $F_1 = 1$, $F_2 = 1$, $F_3 = 2$,.... (5) gives a bound on the cost of the redundancy of our code with $\frac{\gamma}{\alpha} = \frac{2}{(1+\sqrt{5})(\sqrt{2}-1)} \approx 1.492\ldots$.*

Case 5: An example for which there is no known bound.
An interesting open question is how to bound the redundancy for the case of "balanced" words $\mathcal{L}$. i.e., all words which contain exactly as many **0**'s as **1**'s. Note that $\mathcal{L} = \mathcal{D}^*$ where $\mathcal{D}$ is the set of all non-empty balanced words w such that no prefix of w is balanced. Let $d_j = |\{w \in D \mid cost(w) = j\}|$. ¿From standard generating function, we can show $d_j = 0$ for $j = 0$ and odd j and for even $j > 0$, $d_j = 2C_{j/2-1}$ where $C_i = \frac{1}{i+1}\binom{2i}{i}$ is the i^{th} *Catalan number*. Plugging d_j into the characteristic function and using some generating function technique, we show that $c = 1$. But on the other hand, $\sum_{m=1}^{\infty} c_m x^{c_m} = \sum_{j=1}^{\infty} j d_j x^j = 2\sum_{j=1}^{\infty} \binom{2(j-1)}{j-1}(x^2)^j = \frac{x^2}{\sqrt{1-4x^2}}$ *does not* converge when $x = 1/2$. Thus, we can not use Theorem 4 to bound the redundancy. Some observation shows that this $\mathcal{C}$ does not satisfy any of our other theorems either. It remains an open question as to how to construct a code with "small" redundancy for this problem.

7 Open Questions

There are still many open questions left. The first arises by noting that, for the finite case, the previous algorithms were implicitly constructing *alphabetic codes.* Our proof explicitly uses the fact that we are only constructing general codes. It would be interesting to examine whether it is possible to get better bounds for alphabetic codes (or to show that this is not possible).

Another open question is whether it is possible to improve Theorem 4 for some general $\mathcal{C}$ to get a purely additive error rather than a multiplicative one combined with an additive one?

Finally, it would be interesting to devise an analysis that would work for cases where the root exists but $\sum_{m=1}^{\infty} c_m 2^{-cc_m} = \infty$,such as case 5.

References

1. Abrahams, J.: Code and parse trees for lossless source encoding. Communications in Information and Systems 1(2), 113–146 (2001)
2. Altenkamp, D., Melhorn, K.: Codes: Unequal probabilies, unequal letter costs. Journal of the Association for Computing Machinery 27(3), 412–427 (1980)
3. Berger, T., Yeung, R.W.: Optimum '1' ended binary prefix codes. IEEE Transactions on Information Theory 36(6), 1435–1441 (1990)
4. Blachman, N.M.: Minimum cost coding of information. IRE Transactions on Information Theory PGIT-3, 139–149 (1954)
5. Bradford, P., Golin, M., Larmore, L.L., Rytter, W.: Optimal prefix-free codes for unequal letter costs and dynamic programming with the monge property. Journal of Algorithms 42, 277–303 (2002)
6. Capocelli, R.M., De Santis, A., Persiano, G.: Binary prefix codes ending in a "1". IEEE Transactions on Information Theory 40(4), 1296–1302 (1994)
7. Cott, N.: Characterization and Design of Optimal Prefix Codes. PhD Thesis, Stanford University, Department of Computer Science (June 1977)
8. Csisz'ar, I.: Simple proofs of some theorems on noiseless channels. Inform. Contr. 514, 285–298 (1969)
9. Gilbert, E.N.: Coding with digits of unequal costs. IEEE Trans. Inform. Theory 41, 596–600 (1995)
10. Golin, M., Rote, G.: A dynamic programming algorithm for constructing optimal prefix-free codes for unequal letter costs. IEEE Transactions on Information Theory 44(5), 1770–1781 (1998)
11. Golin, M., Jian, L.: More efficient algorithms and analyses for unequal letter cost prefix-free coding (2007), available at `http://arxiv.org/abs/0705.0253`
12. Golin, M.J., Kenyon, C., Young, N.E.: Huffman coding with unequal letter costs. In: Proceedings of the 34th Annual ACM Symposium on Theory of Computing (STOC 2002), pp. 785–791 (2002)
13. Huffman, D.A.: A method for the construction of minimum redundancy codes. In: Proc. IRE 40, vol. 10, pp. 1098–1101 (September 1952)
14. Itai, I.: Optimal alphabetic trees. SIAM J. Computing 5, 9–18 (1976)
15. Karp, R.: Minimum-redundancy coding for the discrete noiseless channel. IRE Transactions on Information Theory IT-7, 27–39 (1961)
16. Krause, R.M.: Channels which transmit letters of unequal duration. Inform. Contr. 5, 13–24 (1962)
17. Marcus, R.S.: Discrete Noiseless Coding. M.S. Thesis, MIT E.E. Dept (1957)
18. Mehlhorn, K.: An efficient algorithm for constructing nearly optimal prefix codes. IEEE Trans. Inform. Theory 26, 513–517 (1980)
19. Patt, Y.N.: Variable length tree structures having minimum average search time. Commun. ACM 12(2), 72–76 (1969)
20. Shannon, C.E.: A mathematical theory of communication. The Bell System Technical Journal 27, 379–423 623–656 (1948)

Kinetic Maintenance of Mobile k-Centres on Trees

Stephane Durocher[1,⋆] and Christophe Paul[2]

[1] School of Computer Science, McGill University, Montréal, Canada
durocher@cs.mcgill.ca
[2] CNRS & LIRMM, Montpellier, France
paul@lirmm.fr

Abstract. Let C denote a set of n mobile clients, each of which follows a continuous trajectory on a weighted tree T. We establish tight bounds on the maximum relative velocity of the 1-centre and 2-centre of C. When each client in C moves with linear motion along a path on T we derive a tight bound of $\Theta(n)$ on the complexity of the motion of the 1-centre and corresponding bounds of $O(n^2\alpha(n))$ and $\Omega(n^2)$ for a 2-centre, where $\alpha(n)$ denotes the inverse Ackermann function. We describe efficient algorithms for calculating the trajectories of the 1-centre and 2-centre of C: the 1-centre can be found in optimal time $O(n \log n)$ when the distance function between mobile clients is known or $O(n^2)$ when the function must be calculated, and a 2-centre can be found in time $O(n^2 \log n)$. These algorithms lend themselves to implementation within the framework of kinetic data structures, resulting in structures that are compact, efficient, responsive, and local.

1 Introduction

Motivation. Finding a set of k points that are central to a collection of data points drawn from a metric space is a fundamental problem of geometry and data analysis. Within the context of facility location, this problem is commonly known as the k-centre problem; given a set P of points (clients) in a metric space S, a k-centre of P is a set of k points (facilities) such that the maximum distance from any client to its nearest facility is minimized. Two common choices for S are a Minkowski distance (typically ℓ_1, ℓ_2, or ℓ_∞) in Euclidean space and graph distance on a weighted graph.

Recently, the k-centre problem has been explored under mobility. In one dimension, the mobile 1-centre problem reduces to maintaining the extrema of a set of mobile clients [1,2,4,16]. Natural generalizations of this problem to higher dimensions in $\mathbb{R}^d$ lead to the mobile Euclidean 1-centre [2,6,10], the mobile rectilinear 1-centre [2,6], and the kinetic convex hull [4,5,16]. Although some mobile k-centre problems can be modelled by motion in Euclidean space, several applications are better represented by motion on a graph. That is, the underlying

⋆ Research for this work was supported by NSERC.

T. Tokuyama (Ed.): ISAAC 2007, LNCS 4835, pp. 341–352, 2007.

graph remains fixed while clients and facilities move along its edges and vertices. Examples include vehicles moving along a road network or mobile robots following defined routes in an industrial setting [7].

Although the static k-centre problem on graphs is well understood, the corresponding mobile problem remained unexplored. Any path in a weighted graph is isometric to a line segment; we generalize the motion of a single client on the line to motion on a path in a graph. That is, given a weighted graph G, each mobile client follows a continuous trajectory along the edges and vertices of G. Continuity and bounded velocity are natural constraints on any physical moving object. It is straightforward to show that for any graph G that contains a cycle, there exist sets of mobile clients on G whose 1-centre is discontinuous. As such, we primarily focus our attention on metric spaces for which the k-centre is continuous. In particular, graph distance on a tree maintains many properties of Euclidean distance in $\mathbb{R}^d$, such as a unique shortest path between two points and a unique, continuous 1-centre, while introducing interesting algorithmic challenges to the problem of maintaining a mobile k-centre.

Main Results. The 1-centre on a tree is unique [18]. We show its motion is continuous and has relative velocity at most one. Since a 2-centre of a tree is not unique, we identify a particular 2-centre which we call the *equidistant 2-centre* and show that its motion is continuous and has relative velocity at most two. The 3-centre is discontinuous even on a line segment; furthermore, no bounded-velocity approximation is possible for the mobile 3-centre [9]. We consider values of k for which the mobile k-centre is continuous: $k \leq 2$.

When each client in C moves with linear motion along a path on T, the motions of the corresponding 1-centre and equidistant 2-centre are piecewise linear. We derive a tight bound of $\Theta(n)$ on the complexity of the motion of the 1-centre, an upper bound of $O(n^2\alpha(n))$ on the complexity of the motion of the equidistant 2-centre, and a worst-case lower bound of $\Omega(n^2)$ on the complexity of the motion of any 2-centre, where $\alpha(n)$ denotes the inverse Ackermann function. We describe efficient algorithms for calculating the trajectories of the 1-centre and 2-centre of C. When the all-pairs distance function between mobile clients is known at all times, the 1-centre can be found in optimal time $O(n \log n)$. The distance function can be calculated in time $O(n^2)$. The equidistant 2-centre can be found in time $O(n^2 \log n)$. Moreover, our algorithms have natural implementations as kinetic data structures (KDS), resulting in structures that are compact, efficient, responsive, and local. Although previous applications of KDSs have been to mobile problems in Euclidean space, as we demonstrate, the KDS framework lends itself naturally to mobile problems on graphs.

2 Definitions

Since a *point* refers to a fixed position in a metric space, we refer to a *client* in the context of motion. Let $C = \{c_1, \ldots, c_n\}$ denote a set of mobile clients, where $I = [0, t_f]$ denotes a time interval, U_T denotes the continuum of points defined

by a weighted tree $T = (V, E)$, and each c_i is a continuous function $c_i : I \to U_T$. For every $t \in I$, let $C(t) = \{c(t) \mid c \in C\}$ denote the set of points in U_T that corresponds to the positions of clients in C at time t. The position of a mobile *facility* f is a function of the positions of a set of clients, $f : \mathscr{P}(U_T) \to U_T$, where $\mathscr{P}(A)$ denotes the power set of set A.

A common assumption in problems involving motion in Euclidean space is that the position of a mobile client is a linear function over time (e.g., [1,2,4]). We make a similar assumption and consider clients with linear motion on trees to establish combinatorial bounds. A mobile client or facility a has *linear motion* if for all $t \in I$, $d(a(0), a(t)) = t \cdot v_a$, where v_a is a non-negative constant and $d(b, c)$ denotes the graph distance between points b and c in U_T. We refer to v_a as the *velocity* of a. The union of the trajectories of a set of n mobile clients that move with linear motion is a subgraph of U_T that has at most $2n$ vertices of degree one. Therefore, we assume that T has at most $2n$ leaves and at most $4n - 1$ vertices, and that $c(0)$ and $c(t_f)$ are vertices of T, for each $c \in C$.

We assume an upper bound of one on the velocity of clients since we are interested in *relative velocity*. Unlike mobile clients, a mobile facility is not required to travel along a path in T nor is its velocity required to remain constant. A mobile facility f has *maximum velocity* v_f if

$$\forall t_1, t_2 \in I,\ d(f(C(t_1)), f(C(t_2))) \leq v_f |t_1 - t_2|, \tag{1}$$

for all sets of mobile clients C defined on any tree T and any time interval I. Continuity is a necessary condition for any fixed upper bound on velocity.

We say client $c \in C$ is *extreme* at time t if $c(t)$ does not lie in the interior of any path through T between two clients in $C(t)$. The *convex hull* of $C(t)$ corresponds to the union of all paths between two clients in $C(t)$.

Definition 1. *Given a weighted tree T and a set of points C in U_T, a* k-centre *of C is a set of k points in U_T, denoted $\Xi_1(C), \ldots, \Xi_k(C)$, that minimizes*

$$\max_{c \in C} \min_{1 \leq i \leq k} d(c, \Xi_i(C)). \tag{2}$$

When $k = 1$, we omit the subscript and write $\Xi(C)$. The definition of a mobile k-centre of a set of mobile clients C follows directly from this static definition.

We refer to (2) as the *k-radius* of C or simply as its *radius* when $k = 1$. The diameter of C is twice the radius of C [19] (for graphs, the diameter is at most twice the radius). A *diametric path* of C is a path between two clients c_1 and c_2 in C such that the distance between them is the diameter of C. We refer to $\{c_1, c_2\}$ as a *diametric pair* and to c_1 and c_2 as *diametric clients*. The 1-centre of C is the unique midpoint of all diametric paths of C [18].

The 1-centre problem on graphs is also known as the absolute centre [18,19], single centre [19], and minimax location problem [8,18]. A common variation of the k-centre problem on graphs is known as the vertex k-centre or discrete k-centre problem, for which the choice of locations for the facility is restricted to vertices (clients) of the graph G. Maintaining continuity in the motion of a mobile facility is impossible in the vertex centre model, as a facility could be required to jump discontinuously from vertex to vertex (client to client).

3 Related Work

Handler [18] gives linear-time algorithms for identifying the 1-centre and 2-centre of a tree. Frederickson gives a linear-time algorithm for finding a k-centre of a tree when k is fixed [13]. Kariv and Hakimi [23] provide an $O(mn+n^2 \log n)$-time algorithm for the 1-centre problem on graphs, where $n = |V|$ and $m = |E|$. See [12,17,20,23,24,25] for reviews of k-centre problems on trees and on graphs.

Kinetic data structures (KDS), introduced by Basch et al. [4], allow the maintenance of an attribute (called the configuration function) of a set of mobile objects moving continuously in some metric space. To do so, a KDS maintains a dynamic set of certificates that guarantees the correctness of the configuration function at any time during the motion. Each certificate c is associated with a small set of mobile objects for which some property is verified. The failure time of certificate c (called an event) is calculated as a function of the motion of these objects. The failure time is added to a priority queue. Restoring the configuration function following a certificate failure requires updating the set of certificates (and the corresponding events in the queue).

Guibas [16] describes four properties used to evaluate the quality of a KDS. A KDS is *compact* if the maximum number of certificates active at any given time is linear in the degrees of freedom of the set of moving objects. A KDS is *responsive* if the maximum number of certificates associated with any one mobile object is polylogarithmic in the problem size. A KDS is *local* if at most a small number of certificates require updating as a result of a certificate failure. A KDS is *efficient* if the total number of certificate failures is proportional to the number of external events (changes to the configuration function). See [3,4,5,16] for a more complete description of the KDS framework.

In relation to our work on the mobile k-centre, KDSs have been constructed to maintain various attributes of a set of mobile clients; these include extremal elements in $\mathbb{R}$ [1,2,4,16], extent (e.g., diameter and width) in $\mathbb{R}^2$ [1,2], approximations of the mobile 1-centre in $\mathbb{R}^2$ [2,6,9,10], approximations of mobile 2-centres in $\mathbb{R}^2$ [11], the kinetic convex hull [4,5,16], an approximation of mobile k-centres in $\mathbb{R}^d$ [15], and approximations of discrete rectilinear k-centres [14,22].

4 The Mobile 1-Centre on Trees

4.1 Properties of the Mobile 1-Centre

The mobile 1-centre is continuous in $\mathbb{R}^d$ [9]. Although the mobile 1-centre has at most unit relative velocity in $\mathbb{R}$, its relative velocity is unbounded in $\mathbb{R}^2$ [6]. It can be shown that the mobile 1-centre is discontinuous on graphs. Restricted to trees, however, we show that the mobile 1-centre remains continuous and has at most unit relative velocity.

Theorem 1. *The mobile 1-centre has relative velocity at most one on trees. This bound is tight.*

Proof. Choose any $t_1, t_2 \in I$ and let $\delta = |t_1 - t_2|$. If $\Xi(C(t_1)) = \Xi(C(t_2))$, then (1) holds trivially. Therefore, assume $\Xi(C(t_1)) \neq \Xi(C(t_2))$. Let P denote the path in T between $\Xi(C(t_1))$ and $\Xi(C(t_2))$. Let r_1 and r_2 denote the respective radii of $C(t_1)$ and $C(t_2)$. Let L_1 denote the subtree of T that includes all branches of $\Xi(C(t_1))$ except P. Note, L_1 includes $\Xi(C(t_1))$. Similarly, let L_2 denote the subtree of T that includes all branches of $\Xi(C(t_2))$ except P.

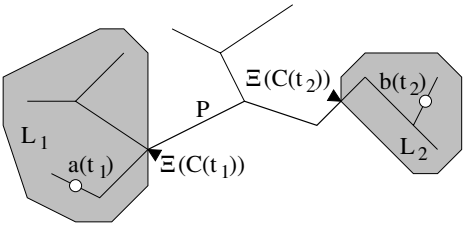


Fig. 1. Illustration in support of Theorem 1

Let a be a client in C such that $a(t_1) \in L_1$ and $d(a(t_1), \Xi(C(t_1))) = r_1$. Similarly, let b be a client in C such that $b(t_2) \in L_2$ and $d(b(t_2), \Xi(C(t_2))) = r_2$. Such clients must exist since $\Xi(C(t))$ is the midpoint of a diametric path of $C(t)$ for all t. See Fig. 1. Therefore,

$$\begin{aligned} d(a(t_1), b(t_2)) &\leq d(a(t_1), \Xi(C(t_1))) + d(\Xi(C(t_1)), b(t_1)) + d(b(t_1), b(t_2)) \\ &\leq 2r_1 + \delta, \qquad (3a) \\ \text{and } d(a(t_1), b(t_2)) &\leq d(a(t_1), a(t_2)) + d(a(t_2), \Xi(C(t_2))) + d(\Xi(C(t_2)), b(t_2)) \\ &\leq 2r_2 + \delta. \qquad (3b) \end{aligned}$$

Consequently,

$$\begin{aligned} & (\ (t_1) \ \ (t_2)) = \ (\ (t_1) \ \ (C(t_1))) + \ (\ (C(t_1)) \ \ (C(t_2))) + \ (\ (C(t_2)) \ \ (t_2)) \\ \Rightarrow \ & (\ (C(t_1)) \ \ (C(t_2))) = \ (\ (t_1) \ \ (t_2)) - \ (\ (t_1) \ \ (C(t_1))) - \ (\ (C(t_2)) \ \ (t_2)) \\ & = \ (\ (t_1) \ \ (t_2)) - \ _1 - \ _2 \\ & \leq \end{aligned}$$

by (3a) and (3b). The bound is realized when the two diametric clients move in a parallel direction. □

It follows that the mobile 1-centre is continuous on trees.

4.2 Complexity of the Motion of the 1-Centre

When n clients move along the real line, each with some constant velocity, the identity of the client that realizes either extremum changes $\Theta(n)$ times in the worst case [4]. In particular, any given client realizes each extremum at most once

in the sequence of changes. When n clients move in $\mathbb{R}^2$ along linear trajectories with constant velocity, the diametric pair of clients changes $\Omega(n^2)$ times in the worst case [1]. As we show in Theorem 2, for a set C of n clients with linear motion on a tree T, the identity of the diametric pair of C changes $\Theta(n)$ times in the worst case. We begin with a definition.

Definition 2. *Given a client c moving with velocity v_c, the* outward velocity *of c at time t, denoted $\vec{v}(c(t))$, is given by*

$$\vec{v}(c(t)) = \begin{cases} -\infty \text{ if } c(t) \text{ is not extreme in } C(t), \\ -v_c \text{ if } c(t) \text{ moves towards the interior of the convex hull of } C(t), \\ v_c \text{ otherwise.} \end{cases}$$

Lemmas 1 through 3 assume linear motion of a set of clients C on a tree T. In addition, we assume that the diameter of C is non-zero at all times; a zero diameter implies that all clients in C coincide in a point and any two clients define a diametric pair. Furthermore, the interior of the convex hull is empty and, consequently, outward velocity is ill defined. We consider a zero diameter in the proof of Theorem 2.

Lemma 1. *The outward velocity of client $c \in C$ is non-decreasing while c remains in a diametric pair of C.*

Proof. Two cases are possible while c remains in a diametric pair of C.

Case 1. Assume c moves away from the interior of the convex hull of C initially. Client c has linear motion along a path $P \subseteq T$. The subpath of P that remains to be travelled by c lies outside the convex hull of C. Therefore the outward velocity of c remains constant.

Case 2. Assume c moves towards the interior of the convex hull of C initially. The outward velocity of c remains constant until c branches and turns away from the interior of the convex hull. The remainder of the motion corresponds to Case 1. □

As we show in Lemma 2, any change in the outward velocity at either endpoint of a diametric path must be increasing.

Lemma 2. *Choose any $t_1 \in I$ and let $\{a_1, b_1\}$ be a diametric pair of $C(t_1)$. If $\{a_2, b_2\}$ is a diametric pair of $C(t_2)$ and a_1 is not in any diametric pair of $C(t_2)$ for some $\epsilon > 0$ and all $t_2 \in (t_1, t_1+\epsilon)$, then $\vec{v}(a_1(t_1)) < \min\{\vec{v}(a_2(t_2)), \vec{v}(b_2(t_2))\}$.*

Proof. Since a_1 is in a diametric pair of $C(t_1)$,

$$\forall c \in C,\ d(a_1(t_1), b_1(t_1)) \geq d(a_1(t_1), c(t_1)). \tag{4}$$

Since $\{a_2, b_2\}$ is a diametric pair of $C(t_2)$ but a_1 is not in any diametric pair of $C(t_2)$,

$$\forall c \in C,\ d(a_1(t_2), c(t_2)) < d(a_2(t_2), b_2(t_2)). \tag{5}$$

Since client motion is continuous and by (4) and (5),

$$d(a_1(t_1), b_1(t_1)) = d(a_2(t_1), b_2(t_1)). \tag{6}$$

The result follows from (5) and (6). □

Lemma 3. *A client $c \in C$ becomes an endpoint of a diametric path of C at most four times.*

Proof. By Definition 2, the outward velocity of a client c in a diametric pair (c is extreme) is one of two values: $\pm v_c$. By Lemma 2, a change in a diametric pair corresponds to an increase in outward velocity. Therefore, a client c realizes either endpoint of a diametric path at most twice, for a total of at most four times. □

Theorem 2. *When each client in C moves with linear motion along a path on T, the motion of the 1-centre of C is piecewise linear and is composed of $\Theta(n)$ linear segments in the worst case, where $n = |C|$.*

Proof. Case 1. Assume the diameter of C is non-zero throughout the motion. The upper bound $O(n)$ follows from Lemma 1 and 3 and the fact that the 1-centre of C is the midpoint of a diametric pair.

Case 2. Assume the diameter of C is zero at some time during the motion. A zero diameter implies that all clients in C coincide at a point; that is, all clients cross simultaneously. This degeneracy occurs at most once since any two clients cross at most once. Since clients in C have linear motion, the motion of the 1-centre of C has linear motion while all clients coincide. Before and after the degeneracy, the motion of clients in C corresponds to Case 1. Therefore, the sum of the number of linear segments of the motion of the 1-centre remains $O(n)$.

The worst-case lower bound of $\Omega(n)$ follows from the corresponding result in one dimension [4]. □

4.3 Kinetic Maintenance of the Mobile 1-Centre

Given a set C of n mobile clients, each moving with linear motion in $\mathbb{R}$, the 1-centre of C is the midpoint of the extrema of C. The position of each extremum is given by the upper (respectively, lower) envelope of the set of n linear functions that correspond to the positions of clients in C relative to a fixed point in $\mathbb{R}$. Hershberger [21] gives an $O(n \log n)$ time algorithm which finds the upper envelope by dividing the set of linear functions in two, recursively finding the upper envelope of each set, and recombining the two envelopes to give the global upper envelope.

Using a related idea, we describe an algorithm for identifying a sequence of diametric pairs of a set of mobile clients, each moving with linear motion on a tree. We then describe how to implement the algorithm as a KDS. The algorithm makes use of the distance function d, where $d(a(t), b(t))$ returns the graph distance on tree T between mobile clients a and b at time t. We begin with the following lemma upon which our algorithm relies.

Lemma 4. *Let C_1 and C_2 be sets of points on U_T for some tree T. Let $\{a_i, b_i\}$ denote a diametric pair of C_i, for $i = 1, 2$. Set $\{e, f\}$ is a diametric pair of $C_1 \cup C_2$, where*

$$\{e, f\} = \operatorname*{argmax}_{\{e', f'\} \subseteq \{a_1, b_1, a_2, b_2\}} d(e', f'). \tag{7}$$

The proof of Lemma 4 was omitted due to space limitations.

Algorithm Description. The set of mobile clients C is partitioned arbitrarily into sets C_1 and C_2 of size $\lfloor n/2 \rfloor$ and $\lceil n/2 \rceil$. For each $i = 1, 2$, the algorithm is called recursively to find a sequence of diametric pairs of C_i, denoted $\{a_{i,1}, b_{i,1}\}, \ldots, \{a_{i,m_i}, b_{i,m_i}\}$, and a corresponding partition of the time interval I, denoted $I_{i,1}, \ldots, I_{i,m_i}$, such that for each j, $a_{i,j}(t)$ and $b_{i,j}(t)$ are a diametric pair of $C_i(t)$ for all $t \in I_{i,j}$. The recursion terminates when $n \leq 2$, in which case each client in C is in a diametric pair. We now describe how to compute a corresponding sequence for C.

Consider a third partition of the time interval I, denoted $I_1, \ldots, I_m$, such that for each i, $I_i = I_{1,j} \cap I_{2,k}$, for some j, k. For all $t \in I_i$, diametric pairs of $C_1(t)$ and $C_2(t)$ consist of four clients in C, say a_1, b_1, a_2, and b_2. Let e and f be defined as in (7). By Lemma 4, e and f are a diametric pair of $C(t)$. The sequence of pairs of clients in $\{a_1, b_1, a_2, b_2\}$ that realize e and f corresponds to the sequence of pairs whose relative distance is maximized. That is, there are six combinations of pairs in $\{a_1, b_1, a_2, b_2\}$, each of which corresponds to an inter-client distance function. The upper envelope of these six functions determines the sequence of identities of e and f during I_i. Thus, solutions to the recursive subproblems are combined to find the sequence of diametric pairs of C.

Time Complexity. By Theorem 2, the complexity of the motion of the 1-centres of C_1 and C_2 is $O(n)$. That is, the time interval I can be partitioned into $O(n)$ subintervals such that the motion of each 1-centre is linear within every subinterval (i.e., $m \in O(n)$). Within each subinterval, we find the maximum of six piecewise-linear functions, each composed of at most four linear segments. Therefore, the maximum function is also piecewise linear, consists of at most 24 linear segments, and can be found in constant time. Thus, the solutions to the two subproblems are combined in $O(n)$ time. The recursion tree has depth $\lfloor \log_2 n \rfloor$, resulting in a total runtime of $O(n \log n)$. The worst-case lower bound of $\Omega(n \log n)$ follows from the corresponding one-dimensional problem [21].

Distance Function. Depending on the formulation of the problem, the input may not include the distance function. In this case, the input is given simply as the set of clients, each of which specifies an origin and destination vertex pair in T. In particular, the path of a client's trajectory is not given.

We assume only a basic weighted edge adjacency list or matrix for the tree T. Build a table $A[i, j]$ that stores the following information for each vertex u_i and each client c_j: $d(u_i, c_j(0))$, the velocity of $c_j(0)$ relative to u_i, and the instant in I (if any) at which the velocity of c_j relative to u_i becomes negated (that is, c_j takes a branch such that its motion changes from towards u_i to away from u_i). This information encodes the two-segment piecewise-linear function $d(u_i, c_j(t))$. Table $A[i, j]$ has size $O(n^2)$ and can be calculated in time $O(n^2)$ by considering each client c_j and tracing its trajectory through T.

For any clients c_1 and c_2 in C, the client-to-client distance function $d(c_1(t), c_2(t))$ can be calculated in constant time from table A. While c_1 and c_2 move towards each other, $d(c_1(t), c_2(t)) = |d(c_1(0), c_1(t)) - d(c_1(0), c_2(t))|$. After one client, say c_1, turns away from the other, $d(c_1(t), c_2(t)) = |d(c_1(t_f), c_1(t)) - d(c_1(t_f), c_2(t))|$.

KDS Implementation. We describe a KDS that maintains a diametric pair over time along with a set of certificates that validates the identity of the pair at any time during the motion.

Theorem 3. *Given a tree T and a set of mobile clients C, each moving with linear motion on a path of T, there exists a KDS to maintain the mobile 1-centre of C that is local, responsive, efficient, and compact.*

Proof. The set of certificates corresponds to the recursive hierarchy described in our algorithm. At any time t, for each set C in the hierarchy, the certificate for $C(t)$ consists of five inequalities that confirm the maximum of six functions. That is, the certificate verifies the identity of a diametric pair of $C(t)$ in terms of the diametric pairs of the subsets $C_1(t)$ and $C_2(t)$ by Lemma 4. The corresponding properties are certified recursively for $C_1(t)$ and $C_2(t)$. Each set maintains a single certificate defined in terms of four clients and the total number of certificates is $O(n)$; therefore, the KDS is compact. Each client is contained in at most $O(\log n)$ sets and, consequently, is associated with at most $O(\log n)$ certificates. As a result, a motion plan update for a client results in changes to the failure times of $O(\log n)$ certificates; therefore, the KDS is local.

A certificate failure occurs whenever the diametric pair of a set C changes. Locally, the certificate for C is restored in constant time; however, a change in the diametric pair of C may percolate upwards in the tree, resulting in $O(\log n)$ additional certificate updates; therefore, the KDS is responsive. By Theorem 2, each set C contributes at most $O(|C|)$ certificate failures, resulting in a total of $O(n \log n)$ certificate failures over the entire motion. Although this value is asymptotically greater than $\Theta(n)$ (the worst-case number of external events for a set of n clients), any offline algorithm for finding the trajectory of the 1-centre requires $\Omega(n \log n)$ time in the worst case, even in one dimension [21]. Therefore, the KDS is efficient. □

5 The Mobile 2-Centre on Trees

5.1 Properties of the Mobile 2-Centre

Although a 2-centre of a set of clients C on a tree is not unique (this is the case even in one dimension [9]), any 2-centre of C, $\Xi_1(C)$ and $\Xi_2(C)$, defines a natural bipartition of C, denoted $\{C_1, C_2\}$, such that

$$\forall c \in C_1,\ d(c, \Xi_1(C)) \leq d(c, \Xi_2(C)) \text{ and } \forall c \in C_2,\ d(c, \Xi_1(C)) \geq d(c, \Xi_2(C)).$$

We refer to $\{C_1, C_2\}$ as a *diametric partition* of C. A diametric partition induced by a given 2-centre is not unique. We refer to the *local 1-centre*, *local radius*, and *local diametric pair/path*, respectively, in reference to the 1-centre, radius, and diametric pair/path of C_1 or C_2. The local 1-centres of C_1 and C_2 are a 2-centre of C [19]. Proofs of results in Sect. 5 were omitted due to space limitations.

5.2 Equidistant 2-Centre

Even in one dimension the motion of a 2-centre defined by two local 1-centres is not continuous. This is easily demonstrated by an example: position a client at each endpoint of a line segment and let a third client move from one endpoint to the other. Not all 2-centres are discontinuous; we describe a strategy for defining the positions of a 2-centre on a tree whose motion is continuous and whose relative velocity is at most two. We refer to this particular 2-centre as the *equidistant 2-centre*:

Definition 3. *Let $\{a, b\}$ be a diametric pair of C. An* equidistant 2-centre *of C, denoted $\{\dot{\Xi}_1(C), \dot{\Xi}_2(C)\}$, is a pair of points that lie on the path between a and b at a distance ρ from a and b, respectively, where ρ denotes the 2-radius of C.*

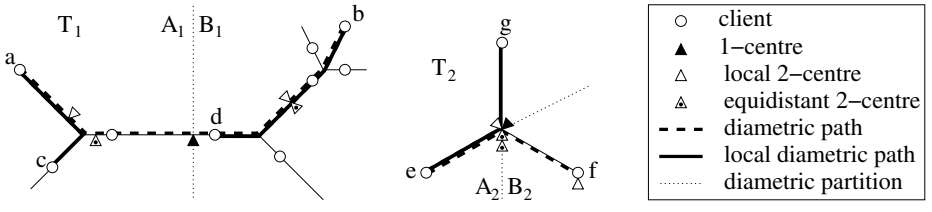


Fig. 2. Equidistant 2-centre examples

See Fig. 2 for an example. It is not difficult to show that the equidistance 2-centre of C is unique and that it is a 2-centre of C. It follows that the equidistant 2-centre is independent of the choice of the diametric pair $\{a, b\}$.

Theorem 4. *Each facility in the mobile equidistant 2-centre has relative velocity at most two.*

It follows that each facility in the mobile equidistant 2-centre is continuous. Since no mobile 2-centre can guarantee relative velocity less than two in one dimension [9], the maximum velocity of the equidistant 2-centre is optimal.

5.3 Complexity of the Motion of the 2-Centre

We establish the following bounds on the complexity of the motion of 2-centres:

Theorem 5. *When each client in C moves with linear motion along a path on T, the motion of each facility in the equidistant 2-centre of C is piecewise linear and is composed of $O(n^2\alpha(n))$ linear segments, where $n = |C|$.*

Theorem 6. *There exists a set of mobile clients C, each moving with linear motion in $\mathbb{R}$, such that the motion of some facility in any 2-centre of C whose motion is piecewise linear is composed of $\Omega(n^2)$ linear segments, where $n = |C|$.*

5.4 Kinetic Maintenance of the Mobile 2-Centre

Capitalizing on our 1-centre results, we describe an algorithm for identifying local 1-centres and the equidistant 2-centre of a set of mobile clients.

Algorithm Description. We first run our 1-centre algorithm to find a sequence of diametric pairs of C, denoted $\{a_1, b_1\}, \ldots, \{a_m, b_m\}$, and a corresponding partition of the time interval I, denoted $I_1, \ldots, I_m$, such that $m \in O(n)$. For each time interval I_i, determine when each client c is closer to a_i and when it is closer to b_i. This determines the sets $C_1(t)$ and $C_2(t)$ for all $t \in I_i$. Consider C_1 (an analogous algorithm applies to C_2). A diametric pair of $C_1(t)$ is given by $a_i(t)$ and a furthest client from $a_i(t)$ in $C_1(t)$. Each local diametric pair determines the motion of the corresponding local 1-centre and the local radius, from which the motion of the equidistant 2-centre is straightforward to calculate.

Time Complexity. For a client $c \in C$, the functions $d(c(t), a_i(t))$ and $d(c(t), b_i(t))$ are piecewise linear, each composed of at most four linear segments. Therefore, c changes partitions $O(1)$ times during interval I_i and calculating the interval for which c resides in either partition is achieved in constant time. Finding a furthest client from $a_i(t)$ for all $t \in I_i$ corresponds to finding the upper envelope of $n-2$ partially-defined, piecewise-linear functions, which can be done in $O(n \log n)$ time using Hershberger's [21] algorithm. Since there are $O(n)$ time intervals, the total runtime is $O(n^2 \log n)$.

Theorem 7. *Given a tree T and a set of mobile clients C, each moving with linear motion on a path of T, there exists a KDS to maintain the mobile equidistant 2-centre of C that is compact and has responsiveness $O(n)$, locality $O(n)$, and efficiency $O(n^2 \log n)$.*

6 Directions for Future Research

As mentioned in Sect. 1, the mobile 1-centre is discontinuous on any cyclic graph. This motivates the search for bounded-velocity approximations of the k-centre on graphs. For the 1-centre, we have preliminary results showing that no continuous $(2-\epsilon)$-approximation is possible for any $\epsilon > 0$. A unit-velocity 2-approximation is given by selecting an arbitrary client $c \in C$ and setting the position of the facility to coincide with $c(t)$. It is unknown whether any bounded-velocity approximation exists for mobile 2-centres on graphs. Finally, it may be possible to extend this work to maintain a discrete 1-centre and 2-centre of C.

Acknowledgements. The authors would like to thank David Kirkpatrick for suggesting that mobile k-centres might be interesting to consider on trees.

References

1. Agarwal, P.K., Guibas, L.J., Hershberger, J., Veach, E.: Maintaining the extent of a moving point set. Disc. & Comp. Geom. 26, 353–374 (2001)
2. Agarwal, P.K., Har-Peled, S.: Maintaining approximate extent measures of moving points. In: SODA, pp. 148–157 (2001)

3. Basch, J.: Kinetic Data Structures. PhD thesis, Stanford U. (1999)
4. Basch, J., Guibas, L., Hershberger, J.: Data structures for mobile data. J. Alg. 31, 1–28 (1999)
5. Basch, J., Guibas, L.J., Silverstein, C., Zhang, L.: A practical evaluation of kinetic data structures. In: SOCG, pp. 388–390 (1997)
6. Bespamyatnikh, S., Bhattacharya, B., Kirkpatrick, D., Segal, M.: Mobile facility location. In: Int. ACM Work. on Disc. Alg. & Meth. for Mob. Comp. & Comm, pp. 46–53 (2000)
7. Bruno, G., Ghiani, G., Improta, G.: Dynamic positioning of idle automated guided vehicles. J. Int. Man. 11, 209–215 (2000)
8. Dearing, P.M., Francis, R.L.: A minimax location problem on a network. Trans. Sci. 8, 333–343 (1974)
9. Durocher, S.: Geometric Facility Location under Continuous Motion. PhD thesis, U. of British Columbia (2006)
10. Durocher, S., Kirkpatrick, D.: The Steiner centre: Stability, eccentricity, and applications to mobile facility location. Int. J. Comp. Geom. & App. 16, 345–371 (2006)
11. Durocher, S., Kirkpatrick, D.: Bounded-velocity approximations of mobile Euclidean 2-centres. Int. J. Comp. Geom. & App. (to appear, 2007)
12. Dvir, D., Handler, G.Y.: The absolute center of a network. Networks 43, 109–118 (2004)
13. Frederickson, G.N.: Parametric search and locating supply centers in trees. In: Dehne, F., Sack, J.-R., Santoro, N. (eds.) WADS 1991. LNCS, vol. 519, pp. 299–319. Springer, Heidelberg (1991)
14. Gao, J., Guibas, L.J., Hershberger, J., Zhang, L., Zhu, A.: Discrete mobile centers. Disc. & Comp. Geom. 30, 45–65 (2003)
15. Gao, J., Guibas, L.J., Nguyen, A.: Deformable spanners and applications. Comp. Geom.: Th. & App. 35, 2–19 (2006)
16. Guibas, L.J.: Kinetic data structures: a state of the art report. In: Work. Alg. Found. Rob, pp. 191–209. A. K. Peters, Ltd (1998)
17. Halpern, J., Maimon, O.: Algorithms for the m-center problems: A survey. Eur. J. Oper. Res. 10, 90–99 (1982)
18. Handler, G.Y.: Minimax location of a facility in an undirected tree graph. Trans. Sci. 7, 287–293 (1973)
19. Handler, G.Y.: Finding two-centers of a tree: The continuous case. Trans. Sci. 12, 93–106 (1978)
20. Hansen, P., Labbé, M., Peeters, D., Thisse, J.F.: Single facility location on networks. An. Disc. Math. 31, 113–146 (1987)
21. Hershberger, J.: Finding the upper envelope of n line segments in $O(n \log n)$ time. Inf. Proc. Let. 33, 169–174 (1989)
22. Hershberger, J.: Smooth kinetic maintenance of clusters. Comp. Geom.: Th. & App. 31, 3–30 (2005)
23. Kariv, O., Hakimi, S.L.: An algorithmic approach to network location problems. I: The p-centers. SIAM J. App. Math. 37, 513–538 (1979)
24. Tansel, B.C., Francis, R.L., Lowe, T.J.: Location on networks: A survey. part I: The p-center and p-median problems. Man. Sci. 29, 482–497 (1983)
25. Tansel, B.C., Francis, R.L., Lowe, T.J.: Location on networks: A survey. part II: Exploiting tree network structure. Man. Sci. 29, 498–511 (1983)

Checking Value-Sensitive Data Structures in Sublinear Space

Michael T. Goodrich[1] and Jonathan Z. Sun[2]

[1] Department of Computer Science, University of California, Irvine, CA 92697-3435
goodrich@ics.uci.edu
[2] School of Computing, The University of Southern Mississippi
118 College Drive, Box 5106, Hattiesburg, MS 39406
jonathan.sun@usm.edu

Abstract. Checking value-sensitive data structures in sublinear space has been an open problem for over a decade. In this paper, we suggest a novel approach to solving it. We show that, in combination with other techniques, a previous method for checking value-insensitive data structures in log space can be extended for checking the more complicated value-sensitive data structures, using log space as well. We present the theoretical model of checking data structures and discuss the types of invasions a checker might bring to the data structure server. We also provide our idea of designing sublinear space checkers for value-sensitive data structures and give a concrete example – a log space checker for the search data structures (SDS).

1 Introduction

Checking the correctness of the computing results of a program with less efforts than recomputing those results finds applications in different areas of computer science. These applications include hardware and software reliability, fault tolerant computing, soundness of algorithms, data authentication, and online transaction auditing, just to name a few. Here we call a program that checks the results of another program and reports errors the *checker*, and the program under check the *checkee*. A checker is evaluated with its validity and efficiency. That is, a checker should be able to catch all the errors and pass all the correct results, and this should be done as efficiently as possible.

A checker is *time efficient* if its time complexity is lower than that of the checkee and the space complexity is not higher than that of the checkee. Similarly, we can define a checker of being *space efficient* when its space complexity is lower and time complexity is not higher. We say that a checker is *optimal* if it has space complexity logarithm of the checkee's space complexity and time complexity $O(1)$ of checking each operation performed by the checkee. (Recall that $O(\log n)$ is the information theoretic lower bound to encode n bits of information.) For example, an optimal search data structure uses $O(n)$ (linear) space and $O(\log n)$ time to do a search, insert, or delete operation.

T. Tokuyama (Ed.): ISAAC 2007, LNCS 4835, pp. 353–364, 2007.

1.1 Theoretic Model

Here we briefly introduce the theoretic model of checking data structures. Most definitions and terminologies follow [1,3,9], but the idea of classifying invasions into two types is new.

The encapsulation model. In this model, users access a data structure D through a set of operations provided by D, for example, $insert(x)$, $delete(x)$, or $search(x)$ if D is a search data structure. The data structure D performs some computation to realize the operation. For some operation, D returns a result to user, like for $search(x)$ it returns a boolean value indicating whether x is contained in D or not. D encapsulates the realization of the operations so that it is irrelevant to the users how and how efficient an operation is fulfilled in D. The checker C is in between of users and D, the checkee, audits the operations issued by users and the results returned by checkee, and reports "error" if any result of an operation is incorrect. This model assumes that D is run on untrusted media with possibility of mistakes or malicious malfunctions, however C is trustworthy. Therefore if the soundness of C is proved, then C and the unreliable D together would encapsulate a reliable D to users. A checker C in this model needs to be individually designed for each data structure D.

Invasions. Checker C is *invasive* if it requires some augmentation of D to facilitate the checking, or if it issues extra operations to D that are not issued by users. The checkers discussed in this paper and in related work are all invasive checkers. (See Section 1.2). We categorize the invasions of those checkers into the following two types:

- *Storage Invasions.* With such invasions, C requests D to associate additional information to each data element stored in D, where the additional information is not necessary for D to perform any operation it provides. As an example, the RAM (random access memory) checker by Blum *et al.* in [3] requests each value x to be stored as a pair (x, i), with i being a discrete index indicating the time (order) of insertion.
- *Operation Invasions.* With such invasions, C issues extra operations to D to follow up a operation issued by a user. Using again the example of Blum *et al.*'s RAM checker [3], the checker C turns each $read(x)$ into a $read(x, i)$ followed by a $write(x, i')$. In another example, the linear space SDS (search data structure) checker by Bright and Sullivan in [5] turns an $insert(x)$ into an $insert(x, i)$ followed by a $predecessor(x)$.

Excessiveness. An invasion is excessive if it increases the asymptotic space complexity of the checkee data structure D to store a data element or the asymptotic time complexity of D to fulfill an original user operation. Otherwise the invasion is non-excessive. A checker with only non-excessive invasions is a non-excessive checker. For example, following up an $insert(x)$ by an operation of $O(n)$ time is an excessive operation invasion to an efficient SDS, which ought to be able to do $insert(x)$ in $O(\log n)$ time. We are only interested in non-excessive checkers.

As two extreme cases, the following invasions are always non-excessive, regardless of the particular time and space complexities of the original unchecked data structure D.

- An storage invasion that requires only $O(1)$ size additional information to be associated to each data element.
- An operation invasion such that the extra operations following each original user operation can be realized in $O(1)$ time by D.

We call such invasions *minimal storage invasions* and *minimal operation invasions*. The checker we provide in this paper commits only minimal (storage and operation) invasions.

Value-insensitive and value-sensitive data structures. Some fundamental data structures have the property that the value stored at each data element plays no role in determining how the data is stored and queried. We call them *value-insensitive data structures*. Maps, arrays, stacks, queues, linked lists, and linked directed graphs are all value-insensitive data structures. [1] When these data structures are queried, the answer is determined solely by the sequence of operations or a specific argument in an operation, such as a memory address, an array index, or a link (pointer). In the opposite, in many advanced data structures such as heaps or binary search trees, the structure to organize data elements and the results of operations depend on a key value contained in each data element. We call these data structures *value-sensitive data structures*. (See [3] for more details.)

Blum et al.'s open problem. We conclude this brief introduction to theoretical model with the following open problem raised by Blum *et al.* in [3]. This paper will suggest and illustrate a novel approach of solving this problem.

Problem 1. Is there any checker under the encapsulation model that checks a value-sensitive data structure such as a binary search tree or heap in sub-linear space? The checker can have only non-excessive storage and operation invasions.

1.2 Related Work

Among the rich literatures on program checking, only a few papers addressed the problem of sublinear space checking. In their fundamental paper [3], Blum *et al.* gave a method of checking unreliable memory (RAM) of size n with a small reliable memory of size $O(\log(n))$, by using ϵ-biased hash functions. In the same paper, authors also applied their method to check two other value-insensitive data structures *stacks* and *queues*. Amato and Loui [1] further extended the result to work on *linked data structures* including lists, trees and general graphs.

[1] Despite its physical meaning, the random access memory (RAM) studied in [3] can be viewed as a map data structure. It maps a memory address to the value stored at this address and supports two operations – writing a value to an address and reading the value from an address.

These checkers all use the ϵ-biased hash functions discovered by J. Naor and M. Naor [14]. Therefore we call them *hash-based* checkers. These checkers commit storage and operation invasions but are extremely efficient. They use $O(\log n))$ space and check each operation in $O(1)$ time w.h.p. However, as pointed out by Blum *et al.* in the open problem, extending this method onto checking value-sensitive data structures is nontrivial.

Independent of Blum *et al.*'s work, some linear space checkers of value-sensitive data structures have been developed using a different technique, which we call the *certificate-based* checking. Sullivan, Wilson and Masson checked the *disjoint set union (DSU)* and a simplified priority queue that doesn't support *delete* or *change_key* in [15]. They also checked *making convex hull (CH)*, *sorting*, and *single-source shortest paths (SSP)* in [16]. Bright and Sullivan checked the full *priority queues (PQ)* supporting *delete* and *change_key* and the *mergeable priority queues (MPQ)* [4]. These certificate-based checkers are offline because they don't report errors immediately but rather maintain a query-result sequence as a certificate trail and verify the trail periodically to find errors. (Note that the hash-based checkers are also offline.) Finkler and Mehlhorn gave a different certificate-based checker for priority queues [9] with the same time and space efficiencies as the one in [4]. This checker is also offline but uses a trail other than the sequence of query-result pairs. Online certificate-based checkers maintain no trail but verify an individual certificate immediately after each operation. Such checkers are developed by Bright and Sullivan for *search data structures (SDS)*, *splittable search data structures (SSDS)*, and *nearest neighbor queries (NN)* [5]. All of the above certificate-based checkers, online or offline, take $O(n)$ (linear) space and $O(1)$ time per operation. The online checkers commit storage and operation invasions, while the offline checkers commit storage invasions only. There was no clue of realizing a certificate-based checker in sublinear space.

1.3 Our Contribution

Inspired by both methods of the hash-based and the certificate-based checking, we suggest a novel approach to checking value-sensitive data structures in sublinear space. We argue that the correctness of value-sensitive data structures consists of two components: *integrity* and *validity*. Then we observe that the hash-based checking technique checks not only the correctness of value-insensitive data structures but also the integrity of value-sensitive data structures. Next, we propose the concept of self-certification of a (value-sensitive) data structure, which is an augmented implementation of the data structure such that the result of an operation is self-certified to guarantee the validity. Using the self-certification and the hash-based techniques together, we realize a checker for SDS (search data structures), a fundamental value-sensitive data structure, which takes $O(\log n)$ space and checks each operation in $O(1)$ time w.h.p. Although the self-certification for SDS is quite simple, this may not be the case for other value-sensitive data structures. Thus, applying our method onto other value-sensitive data structures is non-trivial. However, the frame of our approach does isolate the process of self-certification as the only open part that needs to

Table 1. Comparison of different checking techniques

	Technique	Space	Time	Invasions
1	hash-based	$O(\log(n))$	$O(1)$ w.h.p.	storage, operation
2	online certificate-based	$O(n)$	$O(1)$	storage, operation
3	offline certificate-based	$O(n)$	$O(1)$	storage
4	hash + self-certification*	$O(\log(n))$	$O(1)$ w.h.p.	storage, operation

∗: New in this paper.

Table 2. Data structures that have been checked

	Technique	Applicable to
1	hash-based	RAM, stack, queue, linked structures
2	online certificate-based	SDS, SSDS, NN, PQ
3	offline certificate-based	DSU, PQ, MPQ, CH, sorting, SSP
4	hash + self-certification*	SDS*

∗: New in this paper.

be designed individually for each data structure or each ADT (abstract data type) operation. Therefore, we expect to see more sublinear space checkers of value-sensitive data structures inspired by our work. A comparison of our result and the previous results is shown in Table 1 and 2.

2 Our Idea

2.1 Describing Value-Sensitive Data Structures

We redefine any data structure D as a triple $D = (E, P, R)$, where E, P, R are the set of *elements*, *operations* and *rules*. Here we use the rules to describe all value-sensitive properties of the operations. (Therefore, a value-insensitive data structure is just a data structure with empty set of rules.) For example, a priority queue can be defined as (E, P, R) where

- each data element $e \in E$ stores a key value x;
- P includes operations of $insert(x)$, $delete(x)$, $change\_key(x, x')$, min, and $extract\_min$; and
- R contains one rule: "the element accessed by min or $extract\_min$ has the minimum key value among all key values stored in E".

That is, we consider a data structure as a repository of data elements (a bag of balls) that users can check out and check in elements via some operations following some rules. Observe that the only access or modification an operation can make to E is to check out or check in an element, defined as the following two *basic operations*:

- $put(e)$: check in an element e into E, i.e., change the status from $e \notin E$ to $e \in E$.
- $get(e)$: check out an element e from E, i.e., change the status from $e \in E$ to $e \notin E$.

Any operation is described as the combination of *put* and *get* plus some rules to follow. For example, the *change_key* operation in the above priority queue consists of a $get(e)$ and a $put(e')$; and the *min* operation consists of a $get(e)$ and a $put(e)$ following the rule that e contains the minimum value in E. Note that, as showed with the *min*, even if we just want to take a look at an element, we need to get it from E then put the same element back into E. In order to distinguish the two *basic operations get* and *put* and the original operations provided to users by the checkee, we call the original operations *data operations*.

2.2 Integrity and Validity

The correctness of an operation then bears two meanings: *integrity* and *validity*, which we describe in the following.

- Integrity refers to the authentic execution of each *get* and *put*. That is, a) $e \in E$ is "true" before the execution of $get(e)$ and becomes "false" after it; and b) $e \in E$ is "false" before the execution of $put(e)$ and becomes "true" after it.
- Validity of an operation means that all rules associated to this operation are followed, such as *min* indeed gets the minimum value.

In another word, the integrity is to get or put a ball in the bag honestly, and the validity is to pick the right ball. The data structure D functions correctly if and only if the integrity and the validity are both guaranteed.

2.3 Checking Integrity and Validity Separately

Recall that we define the *put* and *get* as *basic operations* and the original operations supported by the data structure D as *data operations*. Given a data structure D (the checkee), we follow the encapsulation model to design a checker and place it in between of users and D. The checker audits the queries and answers communicated between users and D and reports errors. It maintains a (E, P, R) description of D and checks the integrity and the validity in two separate components.

The validity is checked online. The checker appends additional data operations to a user's data operation and sends them together to the checkee. Based on the results of the additional operations, validity of the user's operation can be verified. Here we must augment D in advance to make it capable of certifying the validity of one data operation with the results of some other data operations. We call this technique the *self-certification* of D, which we will illustrate with SDS in Section 3. Since the validity checking is done online and there is no need to keep a trail, this part of the checker takes constant space. Furthermore, the self-certification of SDS in this paper takes $O(1)$ storage invasions and the additional data operations appended to each user's data operation are done in $O(1)$ time at the checkee. It remains open to do self-certification for other value-sensitive data structures (such as priority queues) with $O(1)$ storage invasions and $O(1)$ time overhead.

In order to check the integrity, the checker transforms every data operation it sends to the checkee (including those of users and those appended by the checker itself) into basic operations and maintains a transcript of the data operations in the form of a sequence of basic operations. It then checks the integrity offline, i.e., to ensure that the basic operations recorded in the transcript are executed honestly at D, by using Blum *et al.*'s hash-based method in [3]. (See Section 3.3.) Since the transcript can be encoded in $O(\log n)$ space using the ϵ-biased hash functions, this part of the checker takes $O(\log n)$ space. Note that a data operation with constant size of input and output will be transformed into the combination of constant number of basic operations, regardless of the running time of that data operation at D. Therefore this integrity checking process brings $O(1)$ time overhead to any data structure that is already self-certificated.

The model of our checking scheme is showed in Figure 1. Next we'll use SDS as a concrete example to demonstrate how to design sublinear space checkers for value-sensitive data structures using our scheme. In Section 3.2 we show how to do self-certification and online validity checking for SDS. This part is not a uniform procedure for all data structures. It must be individually designed for each data structure. For many value-sensitive data structures, it could be a challenging task to do self-certification with non-excessive invasions. This is why our work is not a complete solution to Blum *et al.*'s open problem but only a feasible approach. In Section 3.3 we cite the method provided in [14,3] to show how to check the integrity of a sequence of *put* and *get* operations. This part is uniform for the integrity checking of all data structures. The two parts together give a complete checker for SDS, as well as an approach to checking other value-sensitive data structures.

3 Checking Search Data Structures (SDS) in Log Space

3.1 The Search Data Structures (SDS)

SDS=(E, P, R) is defined as follows. Each element in E contains a key value, and the values are comparable according to a total order. P includes the operations $insert(x)$, $delete(x)$, $search(x)$ $predecesor(x)$, $succesor(x)$, min and max.[2] Rules in R are summarized into two groups in the following.

1. Operation $search(x)$ returns an element $e \in E$, if x is the key value e; or returns "not exist" if x is not the key value of any $e \in E$.
2. Key values of the elements returned by $predecessor(x)$, $successor(x)$, min, and max follow the sorted order of all values stored in E. (Predecessor of the minimum and successor of the maximum are "null".)

[2] Some description to (a succinct version of) SDS contains only three operations $insert(x)$, $delete(x)$, and $search(x)$. However, it is straightforward to augment any implementation of the succinct SDS to support the rest operations in the context by using $O(1)$ size storage invasions and $O(1)$ time operation invasions. Therefore any checker of the (full version) SDS in this paper also checks the succinct version SDS with the same efficiencies and invasions, under Blum *et al.*'s model.

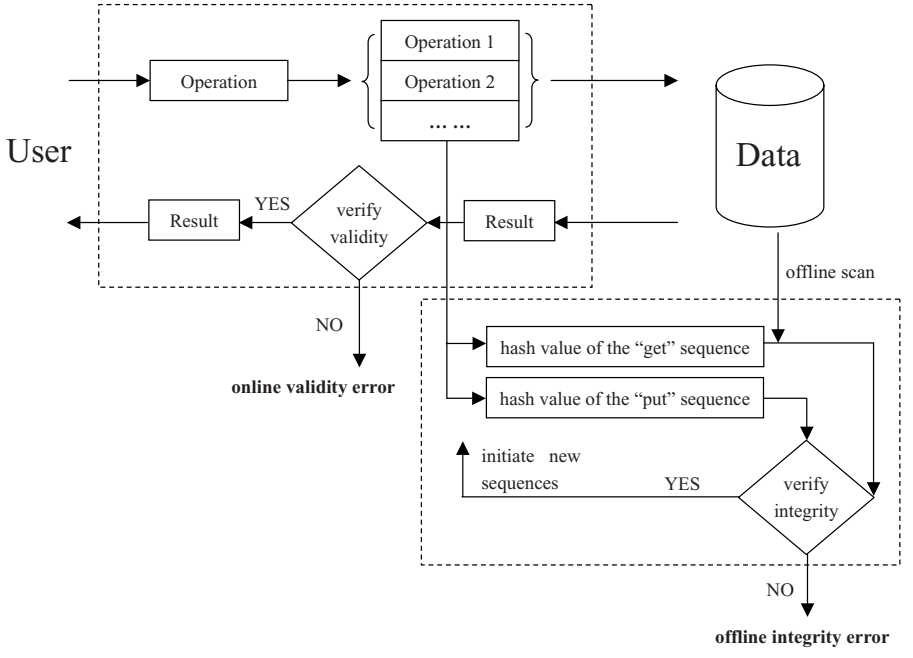


Fig. 1. The scheme of checking value-sensitive data structures

In one word, SDS organizes a set of values in sorted order and supports order preserving queries and updates. An efficient SDS would take $O(n)$ space for storage, $O(\log n)$ time to do each search, insert, or delete operation, and $O(1)$ time to do each of the other operations. A balanced binary search tree with modest augmentations to support a sorted linked list as well (details are omitted) is an example of efficient SDS implementations.

3.2 Self-certification and Validity Checking

We describe this process demonstrated with SDS.

Self-certification. Self-certification of a data structure $D = (E, P, R)$ includes the following three tasks.

1. The augmentation of each data element in E to associate additional information to the element via non-excessive storage invasions. Here for SDS we augment each element $e \in E$ to be $e = (x, predecessor(x), successor(x), i)$, where x is the key value stored in e. Here $predecessor(x)$ and $successor(x)$ associated to e are the key values of (not the links to) the predecessor and successor of x in E according to the sorted order, and i is an integer index indicating the order (discrete time) of insertion of x.

Any data operation that updates the SDS elements, such as an insertion or deletion, must be augmented accordingly as well in order to update the augmented data items consistently. For example, $insert(x)$ now needs to update the elements containing the values $predecessor(x)$ and $successor(x)$ after adding a new element containing x into E. This is like the process of inserting a new node into a linked list. The element containing $predecessor(x)$ must update its successor from $successor(x)$ to x, and the element containing $successor(x)$ must update its predecessor from $predecessor(x)$ to x. We omit the details of augmenting each SDS data operation.

2. A mapping from each data operation in P to a sequence of data operations in P. When auditing the user-checkee correspondence, the checker will substitute the corresponding sequence of operations for each user operations, and use the results of the sequence of operations as a certificate to verify the result of the user operation. Operation invasions caused by this mapping must be non-excessive, meaning that the time complexity of the sequence of operations at D must not exceed that of the original user operation. The mapping for SDS operations is showed in Table 3.
3. Algorithms for the checker to verify the result of each data operation with the results of the sequence of data operations it maps to. This verification checks the validity of the result of a user operation. (I.e., if there is no integrity error, then the operation result follows the rules in R correctly.) The verification algorithms for SDS are showed in the next.

Table 3. The operation mapping of SDS for validity checking

Data operation from user	Sequence of data operations sent to D
$insert(x)$	$predecessor(x), successor(x), insert(x)$
$delete(x)$	$predecessor(x), successor(x), delete(x)$
$search(x)$	$search(x), predecessor(x), successor(x)$
$predecessor(x)$	$predecessor(x), successor(x)$
$successor(x)$	$predecessor(x), successor(x)$
min	min
max	max

Verification algorithms. Algorithms showed below to verify the validity of SDS data operations are straightforward and involve only simple value comparisons. It is easy to justify the soundness of these algorithms, and to see that verifying each data operation takes $O(1)$ time and $O(1)$ space at the checker.

Recall that an element in E with key value x is $e = (x, y, z, i)$ where y and z are the predecessor and successor of x. To assist the presentation, we also denote by (x_1, y_1, z_1, i_1) the element returned by the operation $predecessor(x)$, and (x_2, y_2, z_2, i_2) the element returned by the operation $successor(x)$.

– Certifying $insert(x)$ or $delete(x)$ with $predecessor(x)$ and $successor(x)$. Checker verifies if $z_1 = x_2$, $y_2 = x_1$, and $x_1 < x < x_2$. Certifying $delete(x)$

is similar. Note that if one of the *predecessor*(x) and *successor*(x) is *null*, then verifying the other one still suffices the certification of validity. We omit the details of doing so. We also omit the details when x is already in E before the *insert*(x) or when $x \notin E$ before the *delete*(x), for which cases the verification is still necessary and the process of doing it is similar.

- Certifying *search*(x) with *predecessor*(x) and *successor*(x).
  - If the search result is an element containing x, then the checker verifies if $z_1 = x = y_2$.
  - If it returns "not exist", then the checker verifies if $z_1 = x_2$, $y_2 = x_1$, and $x_1 < x < x_2$.

  Again, if one of the *predecessor*(x) and *successor*(x) is *null*, then verifying the other one is sufficient. Details are omitted.
- Certifying *predecessor*(x) with *successor*(x) or certifying *successor*(x) with *predecessor*(x). Checker verifies either $z_1 = x = y_2$ or ($z_1 = x_2$, $y_2 = x_1$, and $x_1 < x < x_2$). We again omit the cases when *predecessor*(x) or *successor*(x) is *null*.
- Certifying *min* or *max* by itself. Simply verify if the predecessor value of the assumed *min* is *null*, or the successor value of the assumed *max* is *null*.

The above algorithms yield the following result. We omit the proof in this preliminary version.

Theorem 1. *We can check the validity of each SDS data operation in* $O(1)$ *time and* $O(1)$ *space, by introducing storage invasions of* $O(1)$ *size per data element and operation invasions of* $O(1)$ *time per user operation, which are minimal.*

3.3 Checking Integrity with Blum *et al.*'s Method

Properties of ϵ-biased hash functions were studied and fast algorithm to update the hash values was provided by J. Naor and M. Naor in [14]. Its application in checking value-insensitive data structures was discovered by Blum *et al.* in [3]. In fact this technique checks not only the value-insensitive data structures but the integrity of any data structures. (The reason it works solely for checking value-insensitive data structures is that these data structures have empty rules so there is no validity to check.) We present the method in [3] of using hash-based technique to check integrity in the following.

- Checker maintains an integer timer t that increases by 1 after each *put*, which determines the index i of an element $e = (x, y, z, i)$ that is put into E.
- Checker maintains two transcripts W and R to record respectively the sequence of elements that have been put into E and the sequence of elements that have been got from E. Checker transforms every data operation sent to the checkee (including the user's operations and the checker's invasive operations appended to it for validity checking) into the form of *put* and *get*, and appends them to the sequences W and R accordingly.

- If an element e is got out of E and put back later, the time stamp i of it must be updated to the current time t. In another word, the element put back to E is not the one that was got out but a new element. This means that each element is involved in exactly one *put* and one *get* in its life cycle. Elements that are still contained in E have not experienced the *get*.
- Checker periodically scans E to get all elements out. Thus during this period of time the transcripts R for the sequence of *get* and W for the sequence of *put* should be identical. Comparing the two transcripts is sufficient to verify the integrity of the execution of the basic operations *put* and *get* during the past period. (The elements scanned are put back to E after the verification, and those *put* operations are recorded in a new transcript W starting with a new period of integrity checking.)
- Instead of maintaining R and W (each of length $O(n)$) explicitly, the checker maintains the description of an ϵ-biased hash function h and two hash values $h(R)$ and $h(W)$ as fingerprints of R and W. This takes $O(\log n + k)$ space with a constant parameter k. Whenever a new *get* or *put* is appended to R or W, the checker updates $h(R)$ or $h(W)$ accordingly. The amazing properties of ϵ-biased hash functions allow updates to be done bit by bit, without re-hashing the hash value, taking only $O(k)$ time to record a *get* or *put*. All together, the integrity checker uses $O(\log n)$ space and linear time (amortized $O(1)$ time per operation) with a constant parameter k, and reports integrity errors offline (periodically) with probability $1 - 2^{-k}$.

We omit further details of how this method works. Interested readers are referred to read [14,3]. The result of integrity checking is in the following.

Theorem 2. *[14,3] The integrity of any data structure of size $O(n)$ can be checked with probability $1 - 2^{-k}$, k is a constant, by a hash-based checker using $O(\log n)$ space and amortized $O(1)$ time per operation. Such an integrity checker commits $O(1)$ size storage invasions per data element and $O(1)$ time operation invasions per data operation, which are minimal.*

4 Conclusion

We have provided an approach to designing sublinear space checkers for value-sensitive data structures. The framework and the integrity checking component apply for all data structures. The validity checking component must be individually designed for each value-sensitive data structure. We've only showed as a demonstration how to do it on SDS. The two components in Section 3.2 and 3.3 together fulfill a log space, optimal time, and minimal invasion checker for SDS. We conclude with this result in the following theorem, which can be viewed as a step towards a positive answer to the open problem of Blum *et al.* in [3].

Theorem 3. *We can check a search data structure (SDS) with probability $1 - 2^{-k}$, k is a constant, in $O(\log n)$ space and amortized $O(1)$ time per operation, while committing only minimal storage and operation invasions.*

References

1. Amato, N.M., Loui, M.C.: Checking linked data structures. In: FTCS-24: 24th International Symposium on Fault Tolerant Computing, Austin, Texas, pp. 164–175. IEEE Computer Society Press, Los Alamitos (1994)
2. Babcock, B., Babu, S., Datar, M., Motwani, R., Widom, J.: Models and issues in data stream systems. In: Proc. of the 2002 ACM Symp. on Principles of Database Systems (PODS 2002) (2002)
3. Blum, M., Evans, W., Gemmell, P., Kannan, S., Naor, M.: Checking the correctness of memories. Algorithmica 12, 225–244 (1994)
4. Bright, J.D., Sullivan, G.: Checking mergeable priority queues. In: Digest of the 24th Symposium on Fault-Tolerant Computing, 1994, pp. 144–153. IEEE Computer Society Press, Los Alamitos (1994)
5. Bright, J.D., Sullivan, G.: On-line error monitoring for several data structures. In: Digest of the 25th Symposium on Fault-Tolerant Computing, 1995, pp. 392–401. IEEE Computer Society Press, Los Alamitos (1995)
6. Bright, J.D., Sullivan, G., Masson, G.M.: Checking the integrity of trees. In: Digest of the 25th Symposium on Fault-Tolerant Computing, 1995, pp. 402–411. IEEE Computer Society Press, Los Alamitos (1995)
7. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms, 2nd edn. MIT & McGraw-Hill (2001)
8. Devanbu, P., Stubblebine, S.: Stack and queue integrity on hostile platforms. IEEE Tran. Software Engineering 28(1), 100–108 (2002)
9. Finkler, U., Mehlhorn, K.: Checking priority queues. In: Proceedings of the tenth annual ACM-SIAM symposium on Discrete algorithms, Baltimore, Maryland, pp. 901–902 (1999)
10. Goodrich, M.T., Tamassia, R.: Data Structures and Algorithms in Java, 2nd edn. Wiley, Chichester (2001)
11. Kratsch, D., McConnell, R., Mehlhorn, K., Spinrad, J.: Certifying algorithms for recognizing interval graphs. In: SODA 2003, pp. 158–167 (2003)
12. Mehlhorn, K., Näher, S.: The LEDA Platform of Combinatorial and Geometric Computing. Cambridge University Press, Cambridge (1999)
13. Muthukrishnan, S.: Data streams: algorithms and applications. Technical report, Rutger (2003)
14. Naor, J., Naor, M.: Small-bias probability spaces: Efficient constructions and applications. SIAM J. Comput. 22(4), 838–856 (1993)
15. Sullivan, G., Wilson, D., Masson, G.: Certification trails for data structures. In: Proceedings of the 21st Annual Symposium on Fault-Tolerant Computing, pp. 240–247 (1991)
16. Sullivan, G., Wilson, D., Masson, G.M.: Certification of computational results. IEEE Transactions on Computers 44(7), 833–847 (1995)
17. Wasserman, H., Blum, M.: Software reliability via run-time result-checking. Journal of the ACM 44(6), 826–849 (1997)

Manipulation in Games*

Raphael Eidenbenz, Yvonne Anne Oswald,
Stefan Schmid, and Roger Wattenhofer

Computer Engineering and Networks Laboratory
ETH Zurich, Switzerland

Abstract. This paper studies to which extent the social welfare of a game can be influenced by an interested third party within economic reason, i.e., by taking the implementation cost into account. Besides considering classic, benevolent mechanism designers, we also analyze malicious mechanism designers. For instance, this paper shows that a malicious mechanism designer can often corrupt games and worsen the players' situation to a larger extent than the amount of money invested. Surprisingly, no money is needed at all in some cases. We provide algorithms for finding the so-called leverage in games and show that for optimistic mechanism designers, computing the leverage or approximations thereof is **NP**-hard.

1 Introduction

Consider the following extension of the well-known prisoners' dilemma where two bank robbers, both members of the *Al Capone clan*, are arrested by the police. The policemen have insufficient evidence for convicting them of robbing a bank, but they could charge them with a minor crime. Cleverly, the policemen interrogate each suspect separately and offer both of them the same deal. If one testifies to the fact that his accomplice has participated in the bank robbery, they do not charge him for the minor crime. If one robber testifies and the other remains silent, the former goes free and the latter receives a three-year sentence for robbing the bank and a one-year sentence for committing the minor crime. If both betray the other, each of them will get three years for the bank robbery. If both remain silent, the police can convict them for the minor crime only and they get one year each. There is another option, of course, namely to confess to the bank robbery and thus supply the police with evidence to convict both criminals for a four-year sentence (cf. G in Fig. 1). A short game-theoretic analysis shows that a player's best strategy is to testify. Thus, the prisoners will betray each other and both get charged a three-year sentence. Now assume that Mr. Capone gets a chance to take influence on his employees' decisions. Before they take their decision, Mr. Capone calls each of them and promises that if they both remain silent, they will receive money compensating for one year in jail,[1]

* Research supported in part by the Swiss National Science Foundation (SNF). A full version including all proofs, is available as TIK Report 277 at http://www.tik.ee.ethz.ch/.

[1] For this scenario, we presume that time really is money!.

T. Tokuyama (Ed.): ISAAC 2007, LNCS 4835, pp. 365–376, 2007.

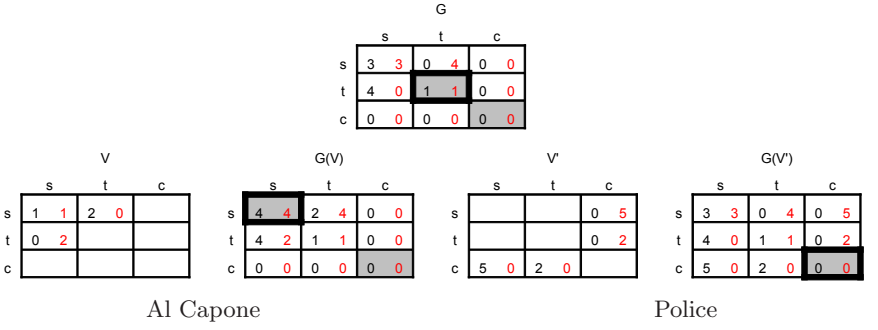


Fig. 1. Extended prisoners' dilemma: G shows the prisoners' initial payoffs, where payoff values equal saved years. The first strategy is to remain silent (s), the second to testify (t) and the third to confess (c). Nash equilibria are colored gray, and non-dominated strategy profiles have a bold border. The left bimatrix V shows Mr. Capone's offered payments which modify G to the game $G(V)$. By offering payments V', the police implements the strategy profile (c, c). As $V_1(c, c) = V_2(c, c) = 0$, payments V' implement (c, c) for free.

and furthermore, if one remains silent and the other betrays him, Mr. Capone will pay the former money worth two years in prison (cf. V in Fig. 1). Thus, Mr. Capone creates a new situation for the two criminals where remaining silent is the most rational behavior. Mr. Capone has saved his clan an accumulated two years in jail.

Let us consider a slightly different scenario where after the police officers have made their offer to the prisoners, their commander-in-chief devises an even more promising plan. He offers each criminal to drop two years of the four-year sentence in case he confesses the bank robbery and his accomplice betrays him. Moreover, if he confesses and the accomplice remains silent they would let him go free and even reward his honesty with a share of the booty (worth going to prison for one year). However, if both suspects confess the robbery, they will spend four years in jail. In this new situation, it is most rational for a prisoner to confess. Consequently, the commander-in-chief implements the best outcome from his point of view without dropping any sentence and he increases the accumulated years in prison by two.

From Mr. Capone's point of view, implementing the outcome where both prisoners keep quiet results in four saved years for the robbers. By subtracting the implementation cost, the equivalent to two years in prison, from the saved years, we see that this implementation yields a benefit of two years for the Capone clan. We say that the *leverage* of the strategy profile where both prisoners play s is two. For the police however, the leverage of the strategy profile where both prisoners play c is two, since the implementation costs nothing and increases the years in prison by two. Since implementing c reduces the players' gain, we say the strategy profile where both play c has a *malicious leverage* of two.

In the described scenario, Mr. Capone and the commander-in-chief solve the optimization problem of finding the game's strategy profile(s) which bear the

largest (malicious) leverage and therewith the problem of implementing the corresponding outcome at optimal cost. This paper analyzes these problems' complexities and presents algorithms for finding the leverage of games for cautious and optimistic mechanism designers. We show that while the leverage of a single strategy profile can be computed efficiently for both cautious and optimistic mechanism designers, finding an optimal implementation for a set of strategy profiles is **NP**-hard by a reduction from the SETCOVER problem, and we provide a lower bound for the approximation attainable by any polynomial-time algorithm. Moreover, we prove that an optimistic mechanism designer cannot compute the leverage of a game in polynomial time unless **P=NP** and finding approximations thereof is hard as well.

Related Work. Algorithmic game theory and mechanism design have become popular tools for gaining insights into the sociological, economical and political complexity of today's distributed systems such as politics, global markets or the Internet (we refer to [6,7] for an introduction). Typically, when a game-theoretic analysis reveals that a system may suffer from selfish behavior, appropriate countermeasures have to be taken in order to enforce a desired behavior (e.g. [4]).

As it is often infeasible for a mechanism designer to influence the rules according to which the players act in a distributed system, she has to resort to other measures. One way of manipulating the players' decision-making is to offer them money for certain outcomes. Monderer and Tennenholtz [5] showed how creditablility can be used to outwit selfish agents and influence their decisions; in some cases no money actually has to be paid at all to implement a certain behavior (cf. also [8]). The authors consider a mechanism designer who cannot enforce behaviors and cannot change the system, and who attempts to lead agents to adopt desired behaviors in a given multi-agent setting. The only way the third party can influence the outcome of the game is by promising non-negative monetary transfers conditioned on the observed behavior of the agents. Eidenbenz et al. [2] have continued the analysis of [5] and have provided deeper insights into the possibilities and algorithmic complexities of mechanism design based on creditability. They presented algorithms for computing a strategy profile set's implementation cost and extended the notion of k-implementation to round-based games, risk-averse player games and average payoff games. Moreover, they show that the complexity results given in [5] are not correct.

This paper extends [2,5] by introducing the concept of *leverage*, a measure for the change of behavior a mechanism design can inflict, taking into account the social gain and the implementation cost. Regarding the payments offered by the mechanism designer as some form of insurance, it seems natural that outcomes of a game can be improved at no costs. However, as a first contribution, in this paper, we show that a malicious mechanism designer can in some cases even *reduce* the social welfare at no costs. Second, we present algorithms to compute both the regular as well as the malicious leverage, and provide evidence that several optimization problems related to the leverage are **NP**-hard.

2 Model

Game Theory. A *strategic game* can be described by a tuple $G = (N, X, U)$. $N = \{1, 2, \ldots, n\}$ is the set of *players* and each Player $i \in N$ can choose a *strategy* (action) from the set X_i. The product of all the individual players' strategies is denoted by $X := X_1 \times X_2 \times \ldots \times X_n$. In the following, a particular outcome $x \in X$ is called *strategy profile* and we refer to the set of all other players' strategies of a given Player i by $X_{-i} = X_1 \times \ldots \times X_{i-1} \times X_{i+1} \times \ldots \times X_n$. An element of X_i is denoted by x_i, and similarly, $x_{-i} \in X_{-i}$; hence x_{-i} is a vector consisting of the strategy profiles available if Player i selects Strategy x_i. $U = (U_1, U_2, \ldots, U_n)$ is an n-tuple of *payoff functions*, where $U_i : X \to \mathbb{R}$ determines Player i's payoff arising from the game's outcome. We will refer to the sum of the individual player's payoffs of a given strategy profile $x \in X$ as the strategy profile's *gain* $U(x) := \sum_{i=1}^{n} U_i(x)$.

Let $x_i, x_i' \in X_i$ be two strategies available to Player i. We say that x_i *dominates* x_i' iff $U_i(x_i, x_{-i}) \geq U_i(x_i', x_{-i})$ for every $x_{-i} \in X_{-i}$ and there exists at least one x_{-i} for which a strict inequality holds. x_i is the *dominant* strategy for Player i if it dominates every other strategy $x_i' \in X_i \backslash \{x_i\}$. x_i is a *non-dominated* strategy if no other strategy dominates it. By $X^* = X_1^* \times \ldots \times X_n^*$ we will denote the set of non-dominated strategy profiles, where X_i^* is the set of non-dominated strategies available to the individual Player i. A strategy profile $x \in X$ is a *Nash equilibrium* if no unilateral deviation in strategy by any single player is profitable, that is $\forall i \in N, U_i(x_i, x_{-i}) \geq U_i(x_i', x_{-i})$.

k-Implementation. We assume that players are rational and always choose a non-dominated strategy. Moreover, they do not cooperate. We examine the impact of payments to players offered by a *mechanism designer* (an interested third party) who seeks to influence the outcome of a game. These payments are described by a tuple of non-negative payoff functions $V = (V_1, V_2, \ldots, V_n)$, where $V_i : X \to \mathbb{R}^+$, i.e. the payments depend on the strategy Player i selects as well as on the choices of all other players. We assume that the players trust the mechanism designer to finally pay the promised amount of money, i.e., consider her trustworthy. The original game $G = (N, X, U)$ is modified to $G(V) := (N, X, [U + V])$ by these payments, where $[U + V]_i(x) = U_i(x) + V_i(x)$, that is, each Player i obtains the payoff of V_i in addition to the payoffs of U_i. The players' choice of strategies changes accordingly: Each player now selects a non-dominated strategy in $G(V)$. Henceforth, the set of non-dominated strategy profiles of $G(V)$ is denoted by $X^*(V)$. For a strategy profile x, the sum of the additional payments to all players is denoted by the *payment* $V(x) := \sum_{i=1}^{n} V_i(x)$. A *strategy profile set* $O \subseteq X$ of G is a subset of all strategy profiles X. Similarly to X_i and X_{-i}, we define $O_i := \{x_i | \exists x_{-i} \in X_{-i} \text{ s.t. } (x_i, x_{-i}) \in O\}$ and $O_{-i} := \{x_{-i} | \exists x_i \in X_i \text{ s.t. } (x_i, x_{-i}) \in O\}$. The mechanism designer's main objective is to force the players to choose a certain strategy profile or a set of strategy profiles, without spending too much. This paper studies two kinds of implementation costs: *worst-case implementation costs* and *uniform implementation costs*.

First, we will consider a pessimistic scenario where the mechanism designer calculates with the maximum possible payments for a desired outcome (*worst-case implementation costs*). For a desired strategy profile set O, we say that payments V *implement* O if $\emptyset \subset X^*(V) \subseteq O$. V is called (worst-case) k-*implementation* if, in addition $V(x) \leq k$, $\forall x \in X^*(V)$. That is, the players' non-dominated strategies are within the desired strategy profile, and the payments do not exceed k for any possible outcome. Moreover, V is an *exact* k-*implementation* of O if $X^*(V) = O$ and $V(x) \leq k$ $\forall x \in X^*(V)$. The *cost* $k(O)$ of implementing O is the lowest of all non-negative numbers q for which there exists a q-implementation. If an implementation meets this lower bound, it is optimal, i.e., V is an *optimal implementation* of O if V implements O and $\max_{x \in X^*(V)} V(x) = k(O)$. The cost $k^*(O)$ of implementing O exactly is the smallest non-negative number q for which there exists an exact q-implementation of O. V is an *optimal exact implementation* of O if it implements O exactly and requires cost $k^*(O)$. The set of all implementations of O will be denoted by $\mathcal{V}(O)$, and the set of all exact implementations of O by $\mathcal{V}^*(O)$. Finally, a strategy profile set $O = \{z\}$ of cardinality one – consisting of only one strategy profile – is called a *singleton*. Clearly, for singletons it holds that non-exact and exact k-implementations are equivalent. For simplicity's sake we often write z instead of $\{z\}$. Observe that only subsets of X which are in $2^{X_1} \times 2^{X_2} \times \ldots \times 2^{X_n}$, i.e., the Cartesian product of subsets of the players' strategies, can be implemented exactly. We call such a subset of X a *convex strategy profile set*.[2] In conclusion, for the worst-case implementation costs, we have the following definitions.

Definition 1 (Worst-Case Cost and Exact Worst-Case Cost). *A strategy profile set O has* worst-case implementation cost $k(O) := \min_{V \in \mathcal{V}(O)}\{\max_{z \in X^*(V)} V(z)\}$. *A strategy profile set O has* exact worst-case implementation cost $k^*(O) := \min_{V \in \mathcal{V}^*(O)}\{\max_{z \in X^*(V)} V(z)\}$.

The assumption that the cost of an implementation V is equal to the cost of the strategy profile in $X^*(V)$ with the *highest* payments is pessimistic. This paper therefore also looks at a less anxious mechanism designer who takes the risk of high worst case costs if the expected costs are small. If players only know their own utilities, assuming them to select one of their non-dominated strategies uniformly at random, is a first simple model an optimistic mechanism designer might apply. We define the uniform cost of an implementation V as the *average* of all strategy profiles' possible cost in $X^*(V)$. Thus we assume all non-dominated strategy profiles $x \in X^*(V)$ to have the same probability.

Definition 2 (Uniform Cost and Exact Uniform Cost). *A strategy profile set O has* uniform implementation cost $k_{UNI}(O) := \min_{V \in \mathcal{V}(O)}\{\varnothing_{z \in X^*(V)} V(z)\}$ *where $\varnothing$ is defined as $\varnothing_{x \in X} f(x) := 1/|X| \cdot \sum_{x \in X} f(x)$. A strategy profile set O has* exact uniform implementation cost $k^*_{UNI}(O) := \min_{V \in \mathcal{V}^*(O)}\{\varnothing_{z \in X^*(V)} V(z)\}$.

[2] These sets define a convex area in the n-dimensional hyper-cuboid, provided that the strategies are depicted such that all o_i are next to each other.

(Malicious) Leverage. Mechanism designers can implement desired outcomes in games at certain costs. This raises the question for which games it makes sense to take influence at all. This paper examines two diametrically opposed kinds of interested parties, the first one being *benevolent* towards the participants of the game, and the other being *malicious*. While the former is interested in increasing a game's social gain, the latter seeks to minimize the players' welfare. We define a measure indicating whether the mechanism of implementation enables them to modify a game in a favorable way such that their gain exceeds the manipulation's cost. We call these measures the *leverage* and *malicious leverage*, respectively. Note that in the following, we will often write "(malicious) leverage" signifying both leverage and malicious leverage.

As the concept of leverage depends on the implementation costs, we examine the *worst-case* and the *uniform* leverage. The worst-case leverage is a lower bound on the mechanism designer's influence: We assume that without the additional payments, the players choose a strategy profile in the original game where the social gain is maximal, while in the modified game, they select a strategy profile among the newly non-dominated profiles where the difference between the social gain and the mechanism designer's cost is minimized. The value of the leverage is given by the net social gain achieved by this implementation minus the amount of money the mechanism designer had to spend. For malicious mechanism designers we have to invert signs and swap max and min. Moreover, the payments made by the mechanism designer have to be subtracted twice, because for a malicious mechanism designer, the money received by the players are considered a loss.

Definition 3 (Worst-Case (Malicious) Leverage). *Let* $lev(O) := \max_{V\in\mathcal{V}(O)}\{\min_{z\in X^*(V)}\{U(z) - V(z)\}\} - \max_{x^*\in X^*} U(x^*)$ *and* $mlev(O) := \min_{x^*\in X^*} U(x^*) - \min_{V\in\mathcal{V}(O)}\{\max_{z\in X^*(V)}\{U(z) + 2V(z)\}\}$. *The* leverage *and* malicious leverage *of a strategy profile set* O *are* $LEV(O) := \max\{0, lev(O)\}$ *and* $MLEV(O) := \max\{0, mlev(O)\}$, *respectively.*

Observe that according to our definitions, leverage values are always non-negative, as a mechanism designer has no incentive to manipulate a game if she will lose money. If the desired set consists only of one strategy profile z, i.e., $O = \{z\}$, we will speak of the *singleton* leverage. Similarly to the (worst-case) leverage, we can define the uniform leverage for less anxious mechanism designers.

Definition 4 (Uniform (Malicious) Leverage). *Let* $lev_{UNI}(O) := \max_{V\in\mathcal{V}(O)}\{\varnothing_{z\in X^*(V)}(U(z) - V(z))\} - \varnothing_{x^*\in X^*} U(x^*)$ *and* $mlev_{UNI}(O) := \varnothing_{x^*\in X^*} U(x^*) - \min_{V\in\mathcal{V}(O)}\{\varnothing_{z\in X^*(V)}\{U(z) + 2V(z)\}\}$. *The* uniform leverage *and* malicious uniform leverage *of a strategy profile set* O *are* $LEV_{UNI}(O) := \max\{0, lev_{UNI}(O)\}$ *and* $MLEV_{UNI}(O) := \max\{0, mlev_{UNI}(O)\}$, *respectively.*

We define the *exact (uniform) leverage* $LEV^*(O)$ and the *exact (uniform) malicious leverage* $MLEV^*(O)$ by simply changing $\mathcal{V}(O)$ to $\mathcal{V}^*(O)$ in the definition of $LEV_{(UNI)}(O)$ and $MLEV_{(UNI)}(O)$. Thus, the exact (uniform) (malicious)

leverage measures a set's leverage if the interested party may only promise payments which implement O exactly.

3 Worst-Case Leverage

Singletons. Consider a mechanism designer seeking to implement a game's best singleton, i.e., the strategy profile with the highest singleton leverage. Dually, a malicious designer attempts to find the profile of the largest malicious leverage.

Due to the fact that $k(z) = \sum_{i=1}^{n} \max_{x_i \in X_i} \{U_i(x_i, z_{-i}) - U_i(z_i, z_{-i})\}$ [5], the leverage of a singleton can be computed efficiently.

Theorem 1. *For a game where every player has at least two strategies, there exists an algorithm which computes all singletons' (malicious) leverage within a strategy profile set O in $\mathrm{O}(n|X|^2)$ time.*

Strategy Profile Sets. Observe that implementing singletons may be optimal for entire strategy sets as well, namely in games where the strategy profile set yielding the largest (malicious) leverage is of cardinality 1. In some games, however, dominating all other strategy profiles in the set is expensive and unnecessary. Therefore, a mechanism designer is bound to consider sets consisting of more than one strategy profile as well to find a subset of X yielding the maximal (malicious) leverage. Moreover, we can construct games where the difference between the best (malicious) set leverage and the best (malicious) singleton leverage gets arbitrarily large. Fig. 2 depicts such a game.

A similar game can be used to show an arbitrarily large difference for the *malicious* leverage: E.g., set the payoffs in the four upper right strategy profiles of the game G in Fig. 2 to 100 instead of 10. V still implements O but switching to O now decreases the social gain.

Although many factors influence a strategy profile set's (malicious) leverage, there are some simple observations. First, if rational players already choose strategies such that the strategy profile with the highest social gain is non-dominated, a designer will not be able to ameliorate the outcome. Just as well, a

$G =$

20	0	11	9	10	10	10	10
11	9	20	0	10	10	10	10
19	10	10	19	9	11	0	20
10	19	19	10	0	20	9	11

$V =$

0	∞	0	∞	0	0	0	0
0	∞	0	∞	0	0	0	0
1	1	1	1	∞	0	∞	0
1	1	1	1	∞	0	∞	0

Fig. 2. Two-player game where set O bears the largest leverage. Implementation V yields $X^*(V) = O$. By offering payments V, a benevolent mechanism designer has cost 2, no matter which $o \in O$ will be played. However, she improves the social welfare by 9. Thus O has a leverage of 7 whereas any singleton $o \in O$ has a leverage of 0. By reducing Player 2's payoffs in the upper game half and Player 1's payoffs in the right game half, O's leverage gets arbitrarily large.

malicious interested party will have nothing to corrupt if a game already yields the lowest social gain possible.

Fact 2. *(i) If a game G's social optimum $x_{opt} := \arg\max_{x \in X} U(x)$ is in X^* then $LEV(G) = 0$. (ii) If a game G's social minimum $x_{worst} := \arg\min_{x \in X} U(x)$ is in X^* then $MLEV(G) = 0$.*

As an example, a class of games where both properties (i) and (ii) of Fact 2 always hold are *equal sum games*, where every strategy profile yields the same gain, $U(x) = c \; \forall x \in X, c : constant$. (Zero sum games are a special case of equal sum games where $c = 0$.)

Fact 3 (Equal Sum Games). *The leverage and the malicious leverage of an equal sum game G is zero: $LEV(G) = 0$, $MLEV(G) = 0$.*

A well-known example of an zero sum game is *Matching Pennies*: Two players toss a penny. If both coins show the same face, Player 2 gives his penny to Player 1; if the pennies do not match, Player 2 gets the pennies. This matching pennies game features another interesting property: There is no dominated strategy. Therefore an interested party could only implement strategy profile sets O which are subsets of X^*. This raises the question whether a set $O \subseteq X^*$ can ever have a (malicious) leverage. We find that the answer is no and moreover:

Theorem 4. *The leverage of a strategy profile set $O \subseteq X$ intersecting with the set of non-dominated strategy profiles X^* is $(M)LEV = 0$.*

In general, the problem of computing a strategy profile set's (malicious) leverage seems computationally hard. It is related to the problem of computing a set's implementation cost, which is conjectured in [2] to be **NP**-hard, and hence, we conjecture the problem of finding $LEV(O)$ or $MLEV(O)$ to be **NP**-hard in general as well. In fact, we can show that computing the (malicious) leverage has at least the same complexity as computing a set's cost.

Algorithm 1 Exact Leverage

Input: Game G, convex set O with $O_{-i} \subset \; _{-i} \forall \; i$
Output: $E \; ^*(O)$

```
1:  _i( ) := 0,   _i( ) := 0 ∀  ∈   , i ∈   ;
2:  _i( _i ¯_i) := ∞ ∀i ∈   ,  _i ∈ O_i , ¯_i ∈  _i\O_i;
3:  compute  _i^*;
4:  return max{0 E   t ev(  n) − max_{x*∈X*}  ( *)};
ExactLev(  , i):
Input: payments  , current Player i
Output: ev*(O) for G( )
1:  if |  _i^*( )\O_i|  0 then
2:     := any strategy in  _i^*( )\O_i;  ev_best := 0;
3:    for all  _i ∈ O_i do
4:      for all  _{-i} ∈ O_{-i} do
5:           _i( _i  _{-i}):=max{0  _i(  _{-i})−
             ( _i( _i  _{-i}) +  _i( _i  _{-i}))};
6:       ev := E   t ev(  +   i);
7:       if ev   ev_best then
8:          ev_best :=  ev;
9:       for all  _{-i} ∈ O_{-i} do
10:          _i( _i  _{-i}) := 0;
11:   return ev_best;
12: if i  1 return E   t ev(  , i − 1);
13: else return min_{o∈O}{ ( ) −  ( )};
```

Theorem 5. *If the computation of a set's implementation cost is* **NP***-hard, then the computation of a strategy profile set's (malicious) leverage is also* **NP***-hard.*

The task of finding a strategy profile set's leverage is computationally hard. Recall that we have to find an implementation V of O which maximizes the term $\min_{z \in X^*(V)}\{U(z) - V(z)\}$. Thus, there is at least one implementation $V \in \mathcal{V}(O)$ bearing O's leverage. Since this V implements a subset of O exactly, it is also

valid to compute O's leverage by searching among all subsets O' of O the one with the largest exact leverage $LEV^*(O')$.[3]

In the following we will provide an algorithm which computes a convex strategy profile set's exact leverage. It makes use of the fact that if $X^*(V)$ has to be a subset of O, each strategy $\bar{o}_i \notin O_i$ must be dominated by at least one strategy o_i in the resulting game $G(V)$ – a property which has been observed and exploited before in [2] in order to compute a set's exact cost. In order to compute $LEV(O)$, we can apply Algorithm 1 for all convex subsets and return the largest value found.

Theorem 6. *Algorithm 1 computes a strategy profile set's exact leverage in time* $\mathrm{O}\big(|X|^2 \max_{i\in N}(|O_i|^{n|X_i^*\setminus O_i|-1}) + n|O| \max_{i\in N}(|O_i|^{n|X_i^*\setminus O_i|})\big)$.

4 Uniform Implementation Cost

We will now turn our attention to the situation of less anxious mechanism designers who anticipate uniform rather than worst-case implementation cost.

Note that for strategy profile sets O with $k^*_{UNI}(O) = 0$ any exact implementation V must have zero payments for any profile inside O, i.e. $V(o) = 0\ \forall o \in O$. Thus, for 0-implementable strategy profile sets, the concepts of worst case exact cost and uniform exact cost coincide, i.e., $k^*_{UNI}(O) = 0$ iff $k^*(O) = 0$. Therefore, Algorithm 2 from [2] decides if O has uniform exact cost of 0 for the uniform case in polynomial as well.

Complexity. In the following we show that it is **NP**-hard to compute the uniform implementation cost for both the non-exact and the exact case. We devise game configurations which reduce SETCOVER to the problem of finding an implementation of a strategy profile set with optimal uniform cost.

Theorem 7. *In games with at least two (three) players, the problem of finding a strategy profile set's exact (non-exact) uniform implementation cost is* **NP***-hard.*

Proof. *Exact Case:* For a given universe $\mathcal{U}$ of l elements $\{e_1, e_2, \ldots, e_l\}$ and m subsets $\mathcal{S} = \{S_1, S_2, \ldots, S_m\}$, with $S_i \subset \mathcal{U}$, SETCOVER is the problem of finding the minimal collection of S_i's which contains each element $e_i \in \mathcal{U}$. We assume without loss of generality that $\nexists (i \neq j) : S_i \subset S_j$. Given a SETCOVER problem instance $SC = (\mathcal{U}, \mathcal{S})$, we can efficiently construct a game $G = (N, X, U)$ where $N = \{1, 2\}$, $X_1 = \{e_1, e_2, \ldots, e_l, s_1, s_2, \ldots, s_m\}$, and $X_2 = \{e_1, e_2, \ldots, e_l, d, r\}$. Each strategy e_j corresponds to an element $e_j \in \mathcal{U}$, and each strategy s_j corresponds to a set S_j. Player 1's payoff function U_1 is defined as follows: $U_1(e_i, e_j) := m+1$ if $i = j$ and 0 otherwise, $U_1(s_i, e_j) := m+1$ if $e_j \in S_i$ and 0 otherwise, $U_1(e_i, d) := 1$, $U_1(s_i, d) := 0$, $U_1(x_1, r) := 0$ $\forall x_1 \in X_1$. Player 2 has a payoff of 0 when playing r and 1 otherwise. In

[3] Note that we do not provide algorithms for computing the malicious leverage but for the benevolent leverage only. However, it is straightforward to adapt our algorithms for the benevolent leverage.

this game, strategies e_j are not dominated for Player 1 because in column d, $U_1(e_j, d) > U_1(s_i, d)$, $\forall i \in \{1, \ldots m\}$. The set O we would like to implement is $\{(x_1, x_2) | x_1 = s_i \wedge (x_2 = e_i \vee x_2 = d)\}$. See Fig. 3 for an example.

Let $Q = \{Q_1, Q_2, \ldots, Q_k\}$, where each Q_j corresponds to an S_i. We now claim that Q is an optimal solution for a SETCOVER problem, an optimal exact implementation V of O in the corresponding game has payments $V_1(s_i, d) := 1$ if $Q_i \in Q$ and 0 otherwise, and all payments $V_1(s_i, e_j)$ equal 0.

Note that by setting $V_1(s_i, d)$ to 1, strategy s_i dominates all strategies e_i which correspond to an element in S_i. Thus, our payment matrix makes all strategies e_i of Player 1 dominated since any strategy e_i representing element e_i is dominated by the strategies s_j corresponding to S_j which cover e_i in the minimal covering set.[4] If there are any strategies s_i dominated by other strategies s_j, we can make them non-dominated by adjusting the payments $V_1(s_i, r)$ for column r. Hence, any solution of SC corresponds to a valid exact implementation of O.

	e_1	e_2	e_3	e_4	e_5	d	r
e_1	5	0	0	0	0	1	0
e_2	0	5	0	0	0	1	0
e_3	0	0	5	0	0	1	0
e_4	0	0	0	5	0	1	0
e_5	0	0	0	0	5	1	0
s_1	5	0	0	5	0	0	0
s_2	0	5	0	5	0	0	0
s_3	0	5	5	0	5	0	0
s_4	5	5	5	0	0	0	0

Fig. 3. Payoff matrix for Player 1 in a game which reduces the SETCOVER problem instance $SC = (\mathcal{U}, \mathcal{S})$ where $\mathcal{U} = \{e_1, e_2, e_3, e_4, e_5\}$, $\mathcal{S} = \{S_1, S_2, S_3, S_4\}$, $S_1 = \{e_1, e_4\}, S_2 = \{e_2, e_4\}, S_3 = \{e_2, e_3, e_5\}, S_4 = \{e_1, e_2, e_3\}$ to the problem of computing $k^*_{UNI}(O)$. The optimal exact implementation V of O in this sample game adds a payment V_1 of 1 to the strategy profiles (s_1, d) and (s_3, d), implying that the two sets S_1 and S_3 cover $\mathcal{U}$ optimally.

It remains to show that such an implementation is indeed optimal and there are no other optimal implementations not corresponding to a minimal covering set. Note that by setting $V_1(s_i, d) := 1$ and $V_1(s_i, r) > 0$ for all s_i, all strategies e_j are guaranteed to be dominated and V implements O exactly with uniform cost $\varnothing_{o \in O} V(o) = m/|O|$. If an implementation had a positive payment for any strategy profile of the form (s_i, e_j), it would cost at least $m + 1$ to have an effect. However, a positive payment greater than m yields larger costs. Thus, an optimal V has positive payments inside set O only in column d. By setting $V_1(s_i, d)$ to 1, s_i dominates the strategies e_j which correspond to the elements in S_i, due to our construction. An optimal implementation has a minimal number of 1s in column d. This can be achieved by selecting those rows s_i ($V_1(s_i, d) := 1$), which form a minimal covering set and as such all strategies e_i of Player 1 are dominated at minimal cost. Our reduction can be generalized for $n > 2$ by simply adding players with only one strategy and zero payoffs in all strategy profiles. We only prove the exact case here. For the non-exact case, we construct a similar

[4] If $|S_j| = 1$, s_j gives only equal payoffs in $G(V)$ to those of e_i in the range of O_2. However, s_j can be made dominating e_i by increasing s_j's payoff $V_1(s_j, r)$ in the extra column r.

game with three players, forcing the mechanism designer to exactly implement O. We refer to the technical report for details. □

Due to the nature of the reduction the inapproximability results of SETCOVER [1,3] carry over to our problem.

Theorem 8. *No polynomial-time algorithm can achieve an approximation ratio better than* $\Omega\left(n \max_i\{\log|X_i^* \setminus O_i|\}\right)$ *for both the exact and non-exact implementation costs within any function of the input length unless* **P**=**NP**.

5 Uniform Leverage

A mechanism designer calculating her average case cost is more optimistic than an anxious designer. Thus, the observation stating that the uniform (malicious) leverage is always at least as large as the worst-case (malicious) leverage does not surprise.

Theorem 9. *A set's uniform (malicious) leverage is always larger or equal the set's (malicious) leverage.*

Another difference concerns the sets O intersecting with X^*, i.e., $O \cap X^* \neq \emptyset$: Unlike the worst-case leverage (Theorem 4), the uniform leverage can exceed zero in these cases, as can be verified by calculating O's leverage in Fig. 3.

Complexity. We conclude our extended abstract with the following theorem on the hardness of computing or approximating the uniform leverage and a proof sketch. We show how the uniform implementation cost can be computed in polynomial time given the corresponding leverage. Thus the complexity of computing the leverage follows from the **NP**-hardness of finding the optimal implementation cost. The lower bounds are derived by modifying the games constructed from the SETCOVER problem in Theorem 7, and by using a lower bound for the approximation quality of the SETCOVER problem. If no polynomial time algorithm can approximate the size of a set cover within a certain factor, we get an arbitrarily small approximated leverage $LEV_{UNI}^{approx} \leq \epsilon$ while the actual leverage is large. Hence the approximation ratio converges to infinity and, unless **P**=**NP**, there exists no polynomial time algorithm approximating the leverage of a game within any function of the input length.

Theorem 10. *For games with at least two (three) players, the problem of computing a strategy profile set's exact (non-exact) uniform (malicious) leverage is* **NP***-hard. Furthermore, the (exact) uniform leverage of* O *cannot be approximated in polynomial time within any function of the input length unless* **P**=**NP**.

Proof (Sketch). **NP**-*Hardness:* For benevolent mechanism designer the claim follows in the exact case from the observation that if $LEV_{UNI}^*(O)$ is found, we can immediately compute $k_{UNI}^*(O)$ which is **NP**-hard (Theorem 7). Due to the fact that any $z \in O$ is also in $X^*(V)$ for any $V \in$

$\mathcal{V}^*(O)$, $lev_{UNI}(O) = \max_{V\in\mathcal{V}^*(O)}\{\varnothing_{z\in X^*(V)}\{U(z) - V(z)\}\} - \varnothing_{z\in X^*} U(x^*) = \max_{V\in\mathcal{V}^*(O)}\{\varnothing_{z\in X^*(V)} U(z) - \varnothing_{z\in X^*(V)} V(z)\} - \varnothing_{x^*\in X^*} U(x^*) = \varnothing_{z\in X^*(V)} U(z) - \min_{V\in\mathcal{V}^*(O)}\{\varnothing_{z\in X^*(V)} V(z)\} - \varnothing_{x^*\in X^*} U(x^*) = \varnothing_{z\in X^*(V)} U(z) - \mathbf{k^*_{UNI}(O)} - \varnothing_{x^*\in X^*} U(x^*)$. Note that $\varnothing_{z\in X^*(V)} U(z)$ and $\varnothing_{x^*\in X^*} U(x^*)$ can be computed in polynomial time. Moreover, it can be shown that it is possible to efficiently construct a problem instance (G', O) from any (G, O) with the same cost, such that for G': $lev_{(UNI)} = LEV_{(UNI)}$. This approach can be applied for the malicious mechanism designers and the non-exact case as well.

Lower Bound Approximation: The game constructed from the SETCOVER problem in Theorem 7 is modified for a benevolent mechanism designer in the exact case as follows: The utilities of Player 1 remain the same. The utilities of Player 2 are all zero except for $U_2(e_1, r) = (l + m)(\sum_{i=1}^{m} |S_i|(m + 1)/(ml + m) - k\mathcal{LB} - \epsilon)$, where k is the minimal number of sets needed to solve the corresponding SETCOVER instance, $\epsilon > 0$, and $\mathcal{LB}$ denotes a lower bound for the approximation quality of the SETCOVER problem. Observe that X^* consists of all strategy profiles of column r. The target set we want to implement exactly is given by $O_1 = \{s_1, ..., s_m\}$ and $O_2 = \{e_1, ..., e_l, d\}$. We compute $lev^{opt}_{UNI} = \varnothing_{o\in O} U(o) - \varnothing_{x\in X^*} U(x) - k = \sum_{i=1}^{m} |S_i|(m + 1)/(ml + m) - \sum_{i=1}^{m} |S_i|(m+1)/(ml+m) - (-k\mathcal{LB} - \epsilon) - k = k(\mathcal{LB} - 1) + \epsilon$. As no polynomial time algorithm can approximate k within a factor $\mathcal{LB}$, $LEV^{approx}_{UNI} \leq \epsilon$. Since $\lim_{\epsilon\to 0} LEV^{opt}_{UNI}/LEV^{approx}_{UNI} = \infty$, the claim follows. A similar modification of the games in the proof of Theorem 7 and corresponding analysis yield the same result for malicious mechanism designers and the non-exact case. □

References

1. Alon, N., Moshkovitz, D., Safra, M.: Algorithmic Construction of Sets for k-Restrictions. ACM Transactions on Algorithms (TALG), 153–177 (2006)
2. Eidenbenz, R., Oswald, Y.A., Schmid, S., Wattenhofer, R.: Mechanism Design by Creditability. In: Dress, A.W.M., Xu, Y., Zhu, B. (eds.) COCOA 2007. LNCS, vol. 4616, Springer, Heidelberg (2007)
3. Feige, U.: A Threshold of $\log n$ for Approximating Set Cover. Journal of the ACM, 634–652 (1998)
4. Feigenbaum, J., Papadimitriou, C., Sami, R., Shenker, S.: A BGP-based Mechanism for Lowest-Cost Routing. In: Proc. 21st Annual ACM Symp. on Principles of Distributed Computing (PODC) (2002)
5. Monderer, D., Tennenholtz, M.: k-Implementation. In: Proc. 4th ACM Conference on Electronic Commerce (EC), pp. 19–28 (2003)
6. Nisan, N., Ronen, A.: Algorithmic Mechanism Design. In: Proc. 31st Annual ACM Symposium on Theory of Computing (STOC) (1999)
7. Osborne, Rubinstein: A Course in Game Theory. MIT Press, Cambridge (1994)
8. Segal, I.: Contracting with Externalities. The Quarterly Journal of Economics, 337–388 (1999)

Using Nash Implementation to Achieve Better Frugality Ratios

Chien-Chung Huang[1], Ming-Yang Kao[2,⋆], Xiang-Yang Li[3,⋆⋆], and Weizhao Wang[3]

[1] Department of Computer Science, Dartmouth College
villars@cs.dartmouth.edu
[2] Department of Electrical Engineering and Computer Science, Northwestern University
kao@cs.northwestern.edu
[3] Department of Computer Science, Illinois Institute of Technology
{xli,wangwei4}@iit.edu

Abstract. Most of the recent works on algorithmic mechanism design exploit the solution concept of dominant strategy equilibria. Such work designs a proper payment scheme so that selfish agents maximize their utility by truthfully revealing their types. It has been pointed out that these truthful mechanisms, the famous among them being the VCG mechanisms, often incur high payments and fruglity ratios. In this work, we exploit the solution concept of Nash implementation to overcome this problem. Our mechanisms induce a set of Nash equilibria so that selfish agents have incentive to act based on a Nash equilibrium. We prove that our mechanisms enjoy substantial advantages over the truthful mechanisms in terms of payment and frugality.

1 Introduction

Algorithmic mechanism design has attracted much attention of computer scientists since the seminal work of Nisan and Ronen [13]. Following their results, most of the works exploit the solution concept of dominant strategy equilibria [1,3,9,10,11,12]. Such work devises a proper payment scheme to ensure that, for each agent, truth-telling will maximize its utility. The VCG mechanism [4,5,15] is probably the most famous representative of the solutions enforcing truth-telling as the dominant strategy. We refer to this class of mechanisms as *truthful mechanisms*. A number of reasons contribute to the popularity of truthful mechanisms, including: (1) relieving each selfish agent from second-guessing whether its declared type is its best choice; and (2) maximizing the social efficiency of the outcome of the game.

In this paper, we introduce a different class of mechanisms which we call *Nash Implementation* mechanisms. Their key difference is that instead of hoping the agents to reveal their types, the proposed mechanisms induce a set of Nash equilibria so that agents maximize their profits by acting based on *any* of the induced Nash equilibria.

⋆ Research supported in part by NSF Grant IIS-0121491.
⋆⋆ Part of the work was conducted when the author was visiting Microsoft Research Asia, Bei-Jing. The research of the author was supported in part by RGC under Grant HKBU 2104/06E and CERG under Grant PolyU-5232/07E.

T. Tokuyama (Ed.): ISAAC 2007, LNCS 4835, pp. 377–389, 2007.

Moreover, given any of the induced equilibria, our mechanisms guarantee that the resultant outcome would be the same as if every agent were telling the truth.

It is noteworthy that, as opposed to our mechanisms, the conventional truthful mechanisms, except the dominant strategy equilibrium, may contain other Nash equilibria which may lead to undesirable outcomes. To demonstrate this possibility, consider the following auction example. Suppose that agents a and b have true types $t_a = 2$ and $t_b = 4$, respectively. The VCG mechanism works by giving the item to the highest bidder (with tie-breaking rule that favors a) and charging him the cost of the second highest bid. As is well-known, the dominant-strategy equilibrium warrants that the item is given to agent b. However, consider the following scenario. Agent a bids 10 and agent b bids 1. This is also a Nash equilibrium, but a gets the item, which is not the desired outcome.

We shall formalize the concept of Nash implementation mechanisms in Section 2. Here we first explain our motivation. Truthful mechanisms aim at soliciting the true types from the agents. Unfortunately, this is often achieved at a high cost. As has been pointed out in [2,8], the VCG mechanism may have to pay $\Theta(n)$ times the cost of the second shortest path in the unicast game. The over-payment phenomenon of truthful mechanisms is more precisely captured by the notion of the frugality ratio [10,14]. For instance, Karlin *et al.* [10] proved certain lower bounds of truthful mechanisms, thus implying that to acquire the true types from agents is often inherently costly.

To circumvent this over-payment problem, we relax the requirement of using the dominant-strategy equilibrium to attain the desired outcome. We allow agents to scheme together and to report their types, which correspond to a Nash equilibrium. The essence of how to Nash implementation mechanisms boils down to how to create a proper inducement so that agents profit by strategizing.

As we will show in the following sections, the most important advantages of Nash implementation mechanisms are their smaller total payments and reduced frugality ratios. Moreover, it is a more stable class of mechanisms in the sense that all Nash equilibria lead to the same desired outcome.

Our Results: To show how Nash implementation mechanisms work, we first present a polynomial-time computable mechanism $\mathcal{M}^{\mathsf{LCPA}}$ for the unicast game. We prove that its total payment is almost always smaller than that of the VCG mechanism; more generally, its payment is only slightly more than the cost of second shortest disjoint path, while the VCG mechanism might pay $\Theta(n)$ times as much. Moreover, $\mathcal{M}^{\mathsf{LCPA}}$ has a frugality ratio $2 + \epsilon$, while any truthful mechanisms has a frugality ratio at least $\Omega(\sqrt{n})$. We note that the frugality ratio of any Nash implementation for the unicast game is at least 2, therefore $\mathcal{M}^{\mathsf{LCPA}}$ is almost optimal.

Considering the more general case of binary demand games, we prove that a necessary condition for the existence of Nash implementation mechanisms is that the social choice function must be monotonic, i.e., a selected agent will still be selected if it declares a smaller cost. This condition turns out to be the same as for the truthful mechanisms. Finally, we present a general framework for designing randomized Nash implementation mechanisms for binary demand games. We prove that our framework yields a frugality ratio comparable to or significantly better than the truthful mechanisms. For example, in the vertex cover game, the frugality ratio of the VCG mechanism is $\Theta(d)$,

where d is the maximum degree in the graph. Our mechanism improves this to $1+\epsilon$, while 1 is clearly the lower bound.

Paper Structure: We review the definitions of mechanisms and introduce the necessary notation in Section 2. Seciton 3 presents a Nash implementation mechanism for the unicast game. Section 4 discusses the frugality ratio of this unicast mechanism. Section 5 gives a general framework of Nash impelmentation for binary game. Section 6 concludes. Due to space constraint, some proofs are omitted here. See full version [6] for details.

2 Preliminaries

A standard economic model for analyzing scenarios in which the agents act according to their own self-interest is as follows. There are n agents. Each agent i, for $i \in \{1, \cdots, n\}$, has some private information $\mathbf{t}_i$, called its *type*. The set of n agents define a type vector $\mathbf{t} = (\mathbf{t}_1, \mathbf{t}_2, \cdots, \mathbf{t}_n)$, which is called the *profile*. An output specification maps each type vector $\mathbf{t}$ to a set of allowed outputs. Agent i's preferences are given by a valuation function v_i that assigns a real number $v_i(\mathbf{t}_i, o)$ to each possible output o. Given the action $\mathbf{a}$ of all agents, the *utility* (often called the *profit*) of agent i is denoted as $\mathbf{u}_i(\mathbf{a}, \mathbf{t}_i)$. For a mechanism $\mathcal{M} = (\mathcal{O}, \mathcal{P})$, we assume that the utility of every agent is quasi-linear, *i.e.*, $\mathbf{u}_i(\mathbf{a}, \mathbf{t}_i) = v_i(\mathbf{t}_i, \mathcal{O}(\mathbf{a})) + \mathcal{P}_i(\mathbf{a})$.

Definition 1. A *mechanism* $\mathcal{M}$ contains two functions: an *output function* $\mathcal{O}$ and a *payment function* $\mathcal{P} = (\mathcal{P}_1, \cdots, \mathcal{P}_n)$*:*

1. For each agent i, it has a set of strategies A_i. It can choose a strategy $a_i \in A_i$.
2. For each strategy vector $a = (a_1, \cdots, a_n)$, i.e., agent i plays strategy $a_i \in A_i$, the mechanism computes an *output* $o = \mathcal{O}(a)$ and a payment $\mathcal{P}(a) = (\mathcal{P}_1(a), \mathcal{P}_2(a), \cdots, \mathcal{P}_n(a))$. The payment $\mathcal{P}_i$ is the money given to each participating agent i. If $\mathcal{P}_i < 0$, it means that the agent has to pay $-\mathcal{P}_i$ to participate in the action.

2.1 Truthful Mechanisms

By the well-known *revelation principle*, we can focus our attention on only the *direct revealing mechanisms*, in which the types are part of the strategy space A_i for each agent i. In practice, it is natural that these mechanisms should satisfy the two properties below:

- **Incentive Compatibility (IC):** Revealing the type $\mathbf{t}_i$ is a *dominant strategy* for each agent i. In other words, for each agent i and any action a, we need that $v_i(\mathbf{t}_i, \mathcal{O}(a|^i\mathbf{t}_i)) + \mathcal{P}_i(a|^i\mathbf{t}_i) \geq v_i(\mathbf{t}_i, \mathcal{O}(a)) + \mathcal{P}_i(a)$. Here, $a|^i\mathbf{t}_i$ denotes that agent i plays strategy $\mathbf{t}_i$ and each of the other agents $j \neq i$ plays strategy a_j.
- **Individual Rationality (IR):** This is also called *Voluntary Participation*. For each agent i and any strategy vector a, it should have $\mathbf{u}_i(a|^i\mathbf{t}_i, \mathbf{t}_i) > 0$, i.e., agent i has a non-negative profit if it reveals its true type $\mathbf{t}_i$.

Definition 2. A direct revealing mechanism is *truthful* (often referred to as *strategy-proof*) if it satisfies IR and IC.

With a truthful mechanism, the agents have no incentive to deviate from the truthful declaration because their overall utility would be no greater than it would have been if they had told the truth. Moreover, the output function guarantees that, given the declared profile $\overline{\mathbf{d}}$ and the actual type vector $\mathbf{t}$, $\mathcal{O}(\mathbf{t}) = \mathcal{O}(\overline{\mathbf{d}})$.

2.2 Nash Implementation Mechanisms

A *Nash implementation* mechanism $\mathcal{M}$ is also composed of a pair of outcome function $\mathcal{O}'$ and a payment method $\mathcal{P}$. It is associated with another social choice function $\mathcal{O}$, which maps a type vector $\mathbf{t}$ to a desirable outcome. The mechanism $\mathcal{M}$ should guarantee that its output function $\mathcal{O}'$ is "faithful" to the social choice function $\mathcal{O}$. We first define what do we mean by "faithful."

Definition 3. *The output function $\mathcal{O}$ and the social choice function $\mathcal{O}'$ have the same* range*, but may not have the same* domain*, $\mathcal{O}'(\overline{\mathbf{d}})$* equals *$\mathcal{O}(\mathbf{t})$, denoted by $\mathcal{O}'(\overline{\mathbf{d}}) \hat{=} \mathcal{O}(\mathbf{t})$, if $\mathcal{O}'_i(\overline{\mathbf{d}}) = \mathcal{O}_i(\mathbf{t})$ for every agent i.*

Note that we allow the two functions $\mathcal{O}$ and $\mathcal{O}'$ to have different domains. Therefore, the declared profile $\overline{\mathbf{d}} = (\overline{\mathbf{d}}_1, \overline{\mathbf{d}}_2, \cdots, \overline{\mathbf{d}}_n)$ could be something different from a declared type vector $\mathbf{t}'$, i.e., the mechanism may require an agent to declare something other than its own type $\mathbf{t}_i$. For example, as we will show in later sections, our mechanisms demand agents to submit *two* bids to join the auction.

Having the above definition, we can now formalize what constitutes a Nash implementation mechanism.

Definition 4 (Nash Implementation Mechanism). Given a social choice function $\mathcal{O}$, a mechanism $\mathcal{M} = (\mathcal{O}', \mathcal{P})$ implements $\mathcal{O}$ in Nash equilibria if*:*

1. $\mathcal{M}$ induces a mapping $\mathbf{T} \rightarrow \overline{\mathbf{D}}$ so that a type vector $\mathbf{t} \in \mathbf{T}$ is mapped to a nonempty subset of declared profiles $\mathcal{M}(t) \subseteq \overline{\mathbf{D}}$.
2. Every declared profile $\overline{\mathbf{d}} \in \mathcal{M}(t)$ is a Nash equilibrium; conversely, every declared profile forming a Nash equilibrium is in the set $\mathcal{M}(t)$.
3. Given any declared profile $\overline{\mathbf{d}} \in \mathcal{M}(t)$, $\mathcal{O}'(\overline{\mathbf{d}}) \hat{=} \mathcal{O}(\mathbf{t})$.

As mentioned earlier, we require that given any type vector, there should exist at least one Nash equilibrium for the declared profile. Moreover, every Nash equilibrium induced by this type vector should ensure that the outcome returned by $\mathcal{O}'$ is the desirable one as if everyone were behaving truthfully under the associated social choice function $\mathcal{O}$.

3 A Nash Implementation Mechanism for Unicast Game

We first review the unicast game. Assume G is a graph representing the network. Every edge e_i corresponds to a selfish agent and has a hidden cost c_i for routing. We need to "buy" a routing path from a source node s to a destination node t. This problem is solvable by the VCG mechanism [13]. Specifically, the VCG mechanism will pick up the least cost path LCP$(s, t, \overline{\mathbf{d}})$ (where $\overline{\mathbf{d}}$ will be identical with the true costs $\mathbf{c}$, as

guaranteed by the mechanism), and pay each chosen edge e_i on $\mathrm{LCP}(s,t,\overline{\mathbf{d}})$ by the amount $\mathcal{P}_k(\overline{\mathbf{d}}) = |\mathrm{LCP}(s,t,\overline{\mathbf{d}}|^k\infty)| - |\mathrm{LCP}(s,t,\overline{\mathbf{d}}|^k 0)|$.

As Archer and Tardos [1] pointed out, the VCG mechanism in the unicast game can be a costly solution. In certain cases, the payment is $\Theta(n)$ times the actual cost of the second shortest path from s to t. To rectify this problem, Immorlica *et al.* [8] proposed the first-price auction mechanism, in which agents are paid whatever the costs they report if they are chosen finally. They point out that a Nash equilibrium may not exist under this mechanism, but strong ϵ-Nash equilibria always do. Moreover, the payment incurred by any strong ϵ-Nash equilibrium is never significantly more, and more often less than that by the VCG mechanism. Our new mechanism is called the *Least Cost Path Auction* (LCPA) mechanism, which is based on the work of Immorlica *et al.* [8].

The mechanism $\mathcal{M}^{\mathsf{LCPA}} = (\mathcal{O}^{\mathsf{LCPA}}, \mathcal{P}^{\mathsf{LCPA}})$ implements the function $\mathcal{O}^{\mathrm{LCP}}$ in Nash equilibria. The social choice function $\mathcal{O}^{\mathrm{LCP}}$ has bid vectors as domain and $\mathcal{O}^{\mathrm{LCP}}(\mathbf{b}) = \mathrm{LCP}(s,t,\mathbf{b})$, returning the shortest path with regard to the bid vector $\mathbf{b}$. The mechanism $\mathcal{M}^{\mathsf{LCPA}}$ requires two bids $\langle \mathbf{b}, \mathbf{b}' \rangle$ from the agents (hence the domain of $\mathcal{O}'$ is composed of two sets of bid vectors); it maps the actual cost $\mathbf{c}$ to a nonempty set of Nash equilibria; moreover, given any $\langle \mathbf{b}, \mathbf{b}' \rangle \in \mathcal{M}^{\mathsf{LCPA}}(\mathbf{c})$, $\mathcal{O}^{\mathsf{LCPA}}(\langle \mathbf{b}, \mathbf{b}' \rangle) = \mathcal{O}^{\mathrm{LCP}}(\mathbf{c})$.

The details about $\mathcal{M}^{\mathsf{LCPA}}$ are given in Algorithm 1. We explain the high-level idea here. First, we compute a shortest path $\mathrm{LCP}(s,t,\mathbf{b})$ based on the first bid $\mathbf{b}$. Then we construct another bid $\mathbf{h}$ so that given any edge i,

$$h_i = \begin{cases} b'_i \text{ if } e_i \in \mathrm{LCP}(s,t,\mathbf{b}); \\ b_i \text{ if } e_i \notin \mathrm{LCP}(s,t,\mathbf{b}). \end{cases} \tag{1}$$

In other words, for those edges which are already on the path $\mathrm{LCP}(s,t,\mathbf{b})$, they can raise their second bid in $\mathbf{b}'$ (but *only to a certain extent*, as will be explained below). Finally, we compute the shortest path $\mathrm{LCP}(s,t,\mathbf{h})$ and pay those chosen edges by the values in $\mathbf{h}$.

To adhere to Nash implementation, our primary concern is to make sure that $\mathrm{LCP}(s,t,\mathbf{h}) = \mathrm{LCP}(s,t,\mathbf{c})$. Our main trick is to seek $\mathrm{LCP}(s,t,\mathbf{c})$ by the agents' first bid $\mathbf{b}$. The difficulty lies in how to ensure that $\mathbf{b} = \mathbf{c}$. Our second concern is how to guarantee that $\mathrm{LCP}(s,t,\mathbf{h})$ returns the same path. As mentioned above, the edges on $\mathrm{LCP}(s,t,\mathbf{b})$ can alter the vector $\mathbf{h}$ by raising their second bid, and they benefit from doing so since they will be paid the amount based on their second bid if they are still chosen in $\mathrm{LCP}(s,t,\mathbf{h})$. Hence, we need to find a way to curb the over-aggressive behavior of the agents in their second bid.

To address the above two concerns, we introduce the following reward-and-punish device. In the beginning, every agent is given a small amount of premium (Line 1). They then will be asked to send a dummy packet with a certain probability (Line 2). We refer to this stage as the *broadcast stage*; this stage can be regarded as dealing out the punishment to any agent who is not giving out its true cost in its first bid. The following lemma captures the reason why we can guarantee that the first bid $\mathbf{b}$ is the same as the actual cost $\mathbf{c}$.

Lemma 1. *For each link e_i, its utility to broadcast $g_i(\mathbf{b}) = -\rho \cdot \mathbf{c}_i + f_i(s,t,\mathbf{b})$ strictly decreases in $[\mathbf{c}_i, +\infty)$ and strictly increases in $(-\infty, \mathbf{c}_i]$ on $\mathbf{b}_i$.*

Algorithm 1. Least Cost Path Auction Routing Mechanism $\mathcal{M}^{\mathsf{LCPA}} = (\mathcal{O}^{\mathsf{LCPA}}, \mathcal{P}^{\mathsf{LCPA}})$

Input: A network $G = (V, E)$, a source $s \in V$, a destination $t \in V$, $\overline{\mathbf{d}} = \langle \mathbf{b}, \mathbf{b}' \rangle$ the declared profile, and two adjustable parameters τ and γ.
Output: A mechanism $\mathcal{M}^{\mathsf{LCPA}}$.
Steps:

1: Set $\mathcal{P}_i^{\mathsf{LCPA}}(\overline{\mathbf{d}}) = f_i(s,t,\mathbf{b})$, where $f_i(s,t,\mathbf{b}) = \tau \cdot \left[b_u \cdot (n \cdot b_u - \sum_{e_j \in G - e_i} b_j) - \frac{b_i^2}{2} \right]$, for each edge $e_i \in G$, and b_u is the maximum cost any edge can declare.
2: With probability $\rho = \tau \cdot (n \cdot b_u - \sum_{e_i \in G} b_i)$, each edge is asked to send a dummy packet.
3: Compute LCP$(s,t,\mathbf{b})$; break ties by lexicographic order. For each edge e_i on LCP$(s,t,\mathbf{b})$, set $\mathbf{h}_i = \mathbf{b}'_i$, set $\mathbf{h}_i = \mathbf{b}_i$ for other edges.
4: Compute LCP$(s,t,\mathbf{h})$ and break ties according to the following rule: the edges on LCP$(s,t,\mathbf{b})$ have the highest priority while the other edges follow the lexicographic order.
5: Set $\mathcal{O}_i^{\mathsf{LCPA}}(\overline{\mathbf{d}}) = 1$ and $\mathcal{P}_i^{\mathsf{LCPA}}(\overline{\mathbf{d}}) = \mathcal{P}_i^{\mathsf{LCPA}}(\overline{\mathbf{d}}) + h_i$ for each edge on LCP$(s,t,\mathbf{h})$; *i.e.* the edges on LCP$(s,t,\mathbf{h})$ will receive the payment and relay the packet.
6: Set $\mathcal{P}_i^{\mathsf{LCPA}}(\overline{\mathbf{d}}) = \mathcal{P}_i^{\mathsf{LCPA}}(\overline{\mathbf{d}}) - \gamma \cdot |b'_i - b_i|$ for each edge in LCP$(s,t,\mathbf{b})$ but not in LCP$(s,t,\mathbf{h})$.

The edges on LCP$(s,t,\mathbf{b})$ can raise their second bid to profit. The key idea is to make sure that they can only raise their bids until a Nash equilibrium is reached. Exceeding this point, the over-bidders will not be chosen in the final path LCP$(s,t,\mathbf{h})$. Moreover, they will be fined a certain amount (Line 6) because of their over aggressive behavior. Those final chosen edges are asked to provide the service and are paid (Line 5). We refer to their actual service ($s-t$ routing) as the *unicast stage*. When computing LCP$(s,t,\mathbf{b})$, we break ties by some lexicographic order (Line 3). But when we compute LCP$(s,t,\mathbf{h})$, we give priority to those edges which are already chosen in LCP$(s,t,\mathbf{b})$. This artifice guarantees the existence of a Nash equilibrium (Line 4)[1].

Nash Equilibria in $\mathcal{M}^{LCPA}$. We now build a set of bid vectors and prove that they contain all the Nash equilibria induced by $\mathcal{M}^{\mathsf{LCPA}}$.

Definition 5. $\langle \mathbf{b}, \mathbf{b}' \rangle$ is said to be a *canonical form* of the bid vectors if*:*

1. $\mathbf{b} = \mathbf{c}$.
2. For each edge $e_i \in$ LCP$(s,t,\mathbf{b})$, $\mathbf{b}_i \le \mathbf{b}'_i \le |\text{LCP}(s,t,\mathbf{c}|^i \infty)| - |\text{LCP}(s,t,\mathbf{c}|^i 0)|$ (which is indeed the payment edge e_i gets under the VCG mechanism).
3. $\sum_{e_i \in LCP(s,t,\mathbf{b})} \mathbf{b}'_i = \sum_{e_j \in SLCP(s,t,\mathbf{c})} \mathbf{c}_j$, where SLCP$(s,t,\mathbf{c})$ is the second shortest $s-t$ path disjoint from LCP$(s,t,\mathbf{c})$.

In Lemma 2 below, we prove that all canonical bid vectors will be Nash equilibria and vice versa in Lemma 3 below. Moreover, canonical bid vectors will lead to the outcome demanded by $\mathcal{M}^{\mathsf{LCPA}}$.

Lemma 2. *(Necessity) A canonical bid vector $\langle \mathbf{c}, \mathbf{c}' \rangle$ is a Nash equilibrium for the mechanism $\mathcal{M}^{LCPA}$. Moreover, such a bid vector guarantees that the final chosen path is correct, i.e., LCP$(s,t,\mathbf{c}) =$ LCP$(s,t,\mathbf{h})$.*

[1] The idea of using the costs of edges to break ties so as to guarantee the existence of a Nash equilibrium is mentioned by Immorlica [7, Page 66].

Lemma 3. *(Sufficiency) Given a pair of bid vectors $\langle \mathbf{b}, \mathbf{b}' \rangle$ which forms a Nash equilibrium, then it must be canonical. Moreover, such a bid vector guarantees that the final chosen path is correct, i.e., LCP$(s,t,\mathbf{c}) =$ LCP$(s,t,\mathbf{h})$.*

Combining Lemma 2 and Lemma 3, we derive the major result of this section:

Theorem 1. *Mechanism $\mathcal{M}^{LCPA}$ implements the social choice function $\mathcal{O}^{LCP}$ in Nash equilibria. The social choice function $\mathcal{O}^{LCP}$ selects the shortest parth from s to t.*

By Definition 5 and Lemmas 2 and 3, the following theorem is immediate.

Theorem 2. *The total payment under the mechanism $\mathcal{M}^{LCPA}$ is at most ϵ more than that of the VCG mechanism, where ϵ can be arbitrarily small. In particular, the total payment is at most ϵ more than the actual cost of the second shortest path disjoint from the shortest path.*

Proof. The edges chosen in LCP$(s,t,\mathbf{h}) =$ LCP$(s,t,\mathbf{c})$ will be paid at most what they would have received under the VCG mechanism, because of the second part of Definition 5. Moreover, by the third part of Definition 5, their collective payment will be at most the cost of the disjoint second shortest path. Additionally, we still have to pay an extra premium $f_i(s,t,\mathbf{b})$ to each agent i. To guarantee the total premium is smaller than ϵ, we set $\tau \leq \epsilon/(n^2 b_u^2)$. □

4 Frugality Ratio in $\mathcal{M}^{\mathsf{LCPA}}$

In this section, we show that the frugality ratio of $\mathcal{M}^{\mathsf{LCPA}}$ is at most $2+\epsilon$, while any Nash implementation mechanism will have frugality at least 2 in the unicast game. Hence $\mathcal{M}^{\mathsf{LCPA}}$ has an almost optimal ratio.

We first review the definition of a frugality ratio. Consider a binary demand game $\mathcal{G} = (\mathscr{E}, \mathscr{F})$, where a set of elements $\mathscr{E}$ comprise the agents, and a certain task can be accomplished by a *feasible* team $f \in \mathscr{F}$ of elements in $\mathscr{E}$. Each element e provides a service and incurs a fixed cost $c_e \in [0,\infty)$ for performing that task. A mechanism $\mathcal{M}$ needs to find a team to perform this task and pay each element in the selected team a certain amount such that certain properties are satisfied, *e.g.*, individual rationality. In [14], Talwar proposed to measure the overpayment for a binary demand game using the *frugality*, which is defined as the total payment of the mechanism (*e.g.*, VCG) over the total cost of the *second optimal team*, which is the best team that does not intersect with the team chosen by the mechanism. The frugality notion was then generalized by Karlin *et al.* [10] to the case where the second optimal disjoint team may not exist. We review their definition here. In a binary demand game with agents $\mathscr{E}$ and feasible sets $\mathscr{F}$, let $\mathbf{T}_{opt}(\mathbf{c})$ be the feasible team with the optimal cost and $\upsilon(\mathbf{c})$ be the solution of the following problem.

$$\upsilon(\mathbf{c}) = \min \sum_{e_i \in \mathbf{T}_{opt}(\mathbf{c})} x_i \quad subject\ to \tag{2}$$

1. $x_i \geq c_i$ for every agent $e_i \in \mathscr{E}$;
2. $\sum_{e_i \in \mathbf{T}_{opt}(\mathbf{c}) - F} x_i \leq \sum_{e_j \in F - \mathbf{T}_{opt}(\mathbf{c})} c_j, \quad \forall F \in \mathscr{F}$; and

3. for every $e_i \in \mathbf{T}_{opt}(\mathbf{c})$, there is a team $F \in \mathscr{F}$ such that $e_i \notin F$ and $\sum_{e_i \in \mathbf{T}_{opt}(\mathbf{c}) - F} x_i = \sum_{e_j \in F - \mathbf{T}_{opt}(\mathbf{c})} c_j$.

Definition 6. *The frugality, denoted by $\phi_{\mathcal{M}}$, of a truthful mechanism $\mathcal{M}$ for a given game $\mathscr{G}$ is defined as $\phi_{\mathcal{M}} = \sup_{\mathbf{c}} \frac{\mathcal{P}(\mathbf{c})}{v(\mathbf{c})}$, i.e., the maximum possible ratio of the total payment by mechanism $\mathcal{M}$ over $v(\mathbf{c})$. The frugality of a game $\mathscr{G} = (\mathscr{E}, \mathscr{F})$ is defined as $\phi_{(\mathscr{E},\mathscr{F})} = \inf_{\mathcal{M}} \phi_{\mathcal{M}}$, where the infimum is taken over all truthful mechanisms $\mathcal{M}$.*

We can extend the frugality definition for truthful mechanisms to Nash implementation truthful mechanisms in a natural way as follows.

Definition 7. *The frugality, denoted by $\phi_{\mathcal{M}}$, of a Nash implementation mechanism $\mathcal{M}$ for a given game $\mathscr{G}$ is defined as $\phi_{\mathcal{M}} = \sup_{\mathbf{c}} \frac{\mathcal{P}(\overline{\mathbf{d}})}{v(\mathbf{c})}$, i.e., the maximum possible ratio of the total payment of the mechanism $\mathcal{M}$ over $v(\mathbf{c})$. Here $\overline{\mathbf{d}}$ is a Nash equilibrium under $\mathcal{M}$ when the true cost vector of agents is $\mathbf{c}$. The frugality of a game $\mathscr{G} = (\mathscr{E}, \mathscr{F})$ based on output method $\mathcal{O}$ is defined as $\phi_{(\mathscr{E},\mathscr{F},\mathcal{O})} = \inf_{\mathcal{M}} \phi_{\mathcal{M}}$, where the infimum is taken over all* Nash implementation truthful *mechanisms $\mathcal{M}$ with respect to method $\mathcal{O}$. The frugality of a game $\mathscr{G} = (\mathscr{E}, \mathscr{F})$ is defined as $\phi_{(\mathscr{E},\mathscr{F})} = \inf_{\mathcal{M}} \phi_{\mathcal{M}}$, where the infimum is taken over all Nash implementation mechanisms $\mathcal{M}$.*

We show below that $\mathcal{M}^{\mathsf{LCPA}}$ achieves frugality $2 + \epsilon$, and this ratio is almost optimal. Before we proceed, we should point out that Karlin *et al.* [10] proved that the VCG mechanism has frugality $\Theta(n)$ and *any* truthful mechanism for the unicast game has frugality $\Omega(\sqrt{n})$. Clearly, Nash implementation mechanisms have better frugality.

Theorem 3. *The frugality of the mechanism $\mathcal{M}^{\mathsf{LCPA}}$ is $2 + \epsilon$, and the frugality of any Nash implementation mechanism based on LCP is at least 2.*

5 Nash Implementation Mechanisms for Binary Demand Games

In this section, we give a general framework for a Nash implementation mechanism. Assuming a social choice function $\mathcal{O}^{opt}$ which returns the team with minimum cost, Algorithm 2 gives a mechanism implements $\mathcal{O}^{opt}$ in Nash equilibria. Without loss of generality, we assume that the given set system $(\mathscr{E}, \mathscr{F})$ is *upwards closed*, i.e. for every $S \in \mathscr{F}$ and every superset T with $S \subseteq T \subseteq \mathscr{E}$, we have $T \in \mathscr{F}$. To avoid triviality, we assume that the system is *monopoly-free*, i.e., there is no element present in all feasible teams. For any set $T \subseteq \mathscr{E}$, let $w(T, \mathbf{c}) = \sum_{e \in T} c_e$ be the weight of the team T under cost vector $\mathbf{c}$.

The intuition for Algorithm 2 is similar to the Algorithm 1. In the beginning, every agent is given a premium (Line 1). There is a chance that it will be recruited into the team (Line 2) even if it does not belong to the optimal team. We make the final decision based on $\mathbf{h}$ and punish the over-greedy bidders (Line 6).

Theorem 4. *With probability $1 - \epsilon$, for arbitrarily small ϵ, the mechanism $\mathcal{M}^{out}$ implements $\mathcal{O}^{opt}$ in Nash equilibria.*

Algorithm 2. General Framework to Design Nash Implementation Mechanisms for Binary Demand Game

Input: A set system $(\mathscr{E}, \mathscr{F})$, $\overline{\mathbf{d}} = \langle \mathbf{b}, \mathbf{b}' \rangle$ the declared profile, and two adjustable parameters τ and γ.
Output: A mechanism implementing $\mathcal{M}^{out} in Nash equilibria$.
Steps:

1: Set $\mathcal{P}_i^{out}(\overline{\mathbf{d}}) = f_i(s,t,\mathbf{b})$, where $f_i(\mathbf{b}) = \tau \cdot \left[b_u \cdot (n \cdot b_u - \sum_{e_j \in \mathscr{E} - e_i} b_j) - \frac{b_i^2}{2} \right]$ for each agent i.
2: With probability $\rho = \tau \cdot (n \cdot b_u - \sum_{e_i \in \mathscr{E}} b_i)$, we select all elements and ask them to perform the service.
3: Find the optimal teams and break ties by favoring teams with bigger sizes. After that, break ties by lexicographic order. For every agent i on $T_{opt}(\mathbf{b})$, set $\mathbf{h}_i = \mathbf{b}'_i$; otherwise set $\mathbf{h}_i = \mathbf{b}_i$.
4: Find the optimal teams on $T_{opt}(\mathbf{h})$ and break ties according to the rule that teams containing members in $T_{opt}(\mathbf{b})$ have the highest priority and if two teams have the same cost, choose the one that contains more agents in $T_{opt}(\mathbf{b})$.
5: Set $\mathcal{O}_i^{out}(\overline{\mathbf{d}}) = 1$ and $\mathcal{P}_i^{out}(\overline{\mathbf{d}}) = \mathcal{P}_i^{out}(\overline{\mathbf{d}}) + \mathbf{h}_i$ for each agent in $T_{opt}(\mathbf{h})$.
6: Set $\mathcal{P}_i^{out}(\overline{\mathbf{d}}) = \mathcal{P}_i^{out}(\overline{\mathbf{d}}) - \gamma \cdot |b'_i - b_i|$ for each agent on $T_{opt}(\mathbf{b}) - T_{opt}(\mathbf{h})$.

The proof can be obtained through Lemmas 4 and 5 below. Here we only mention that there is a small probability that the final recruited team is a superset of the optimal team. To make sure the probability of this happening is smaller than ϵ, we make $\tau \leq \epsilon/(n^2 b_u)$.

Similar to the mechanism $\mathcal{M}^{\mathsf{LCPA}}$ for unicast game, any Nash equilibrium for the mechanism $\mathcal{M}^{out}$ has the following properties.

Lemma 4. *If $\overline{\mathbf{d}} = \langle \mathbf{b}, \mathbf{b}' \rangle$ is a Nash equilibrium, then (1) $\mathbf{b} = \mathbf{c}$; (2) $\mathbf{b}'_i = \mathbf{c}_i$ if $i \notin T_{opt}(\mathbf{c})$ and $\mathbf{b}'_i \geq \mathbf{c}_i$ otherwise; (3) for any feasible team T, $w(T, \mathbf{b}') \geq w(T_{opt}(\mathbf{c}), \mathbf{b}')$.*

Before we present our main results in Theorem 5 below, we first introduce the notion of the worst and best Nash equilibria for binary demand games. Consider the linear program (2); interestingly, each solution that satisfies all constraints corresponds to a subvector of $\mathbf{b}'$ in a Nash equilibrium $\overline{\mathbf{d}}$.

Lemma 5. *There exists an one-to-one mapping between the feasible solution satisfying the constraints of linear program (2) and $\mathbf{b}'$ of a Nash equilibrium $\overline{\mathbf{d}}$ for the mechanism $\mathcal{M}^{out}$.*

Recall that the main part of payment is the sum of the bids in the subvector. Thus, for convenience of presentation, we call each solution that meets all constraints of the linear program (2) a *NE bid*. Thus, the solution to linear program (2) is the *minimum NE bid*, denoted by $\mathbf{X}^{\min}$. Next, we introduce a counterpart of the linear program (2). Let $\varrho(\mathbf{c})$ be the solution of the following linear program:

$$\varrho(\mathbf{c}) = \max \sum_{e_i \in T_{opt}} x_i \quad subject\ to \tag{3}$$

1. $x_i \geq c_i$ for each agent $e_i \in \mathscr{E}$;

2. $\sum_{e_i \in T_{opt}(\mathbf{c})-T} x_i \leq \sum_{e_j \in T-T_{opt}(\mathbf{c})} c_j, \quad \forall T \in \mathscr{F}$; and
3. For every $e_i \in T_{opt}(\mathbf{c})$, there is a $T \in \mathscr{F}$ such that $e_i \notin T$ and $\sum_{e_i \in T_{opt}(\mathbf{c})-T} x_i = \sum_{e_j \in T-T_{opt}(\mathbf{c})} c_j$.

The solution to linear program (3) is *maximum NE bids*, denoted by $\mathbf{X}^{\max}$. The ratio of the maximum and minimum NE bids is $\lambda = \frac{\sum_{e_i \in T_{opt}(\mathbf{c})} x_i^{\max}}{\sum_{e_i \in T_{opt}(\mathbf{c})} x_i^{\min}}$, which is called *NE ratio*. Interestingly, NE ratio always equals the ratio between the *price of stability* over the *price of the anarchy*. Let $OPT(\mathbf{c})$ be the globally optimum solution. Recall that the price of stability is defined as $\frac{\varrho(\mathbf{c})}{OPT(\mathbf{c})}$, i.e., the ratio of the value of the largest Nash equilibrium over the optimum solution. The price of anarchy is defined as $\frac{\upsilon(\mathbf{c})}{OPT(\mathbf{c})}$, i.e., the ratio of the value of the smallest Nash equilibrium over the optimum solution. Next, we show a relationship between the Nash equilibrium and the frugality of a Nash implementation mechanisms.

Theorem 5. *Let $\mathcal{M}^{out} = (\mathcal{O}', \mathcal{P})$ be the mechanism computed by Algorithm 2. Then its frugality is $\lambda + \epsilon$, where λ is the NE ratio and ϵ is a positive number depending on parameters τ and γ.*

Proof. From Lemma 5, if $\overline{\mathbf{d}}$ is a Nash equilibrium of $\mathcal{M}^{out}$, then $\mathbf{b}'$ satisfies all the constraints of linear program (3). Thus, $\sum_{e_i \in T_{opt}(\mathbf{c})} b_i' \leq \sum_{e_i \in T_{opt}(\mathbf{c})} x_i^{\max}$. Recall that the total payment is

$$\begin{aligned}\mathbb{P}(\overline{\mathbf{d}}) &= \sum_{e_i} f_i(\mathbf{b}) + \sum_{e_i \in T_{opt}(\mathbf{c})} b_i' \leq \sum_{e_i \in T_{opt}(\mathbf{c})} x_i^{\max} + \tau \cdot n^2 b_u^2 \\ &= (1+\epsilon) \cdot \sum_{e_i \in T_{opt}(\mathbf{c})} x_i^{\max} = (1+\epsilon) \cdot \lambda \cdot \sum_{e_i \in T_{opt}(\mathbf{c})} x_i^{\min}.\end{aligned}$$

This proves the theorem. □

Recall that in order to compute the NE ratio λ, we need to solve linear program (2), which is still an open problem even for some special binary demand games, *e.g.*, the shortest path game. However, we are able to get tight bounds of the NE ratio for some binary demand games using different approaches. Before we show how to bound the NE ratio, we define some terms first.

We call a feasible team T a *base-team* if removing any elements from T makes it unfeasible. Given a team $T_1 \subseteq T$ where T is a base team, we say that team T_2 *covers* T_1 through T if $(T - T_1) \bigcup T_2$ is also a feasible team. T_2 covers an element e_i through T if there exists an $T_1 \subseteq T$ such that $e_i \in T_1$ and T_2 covers T_1 through T.

A team set $\mathscr{T}$ is a *team-cover* of a base-team T if for each element $e_i \in T$, there exists a team $T_i \in \mathscr{T}$ such that T_i covers e_i through T. A team cover $\mathscr{T}$ of T is a *minimal team cover* (MTC) of T if (1) $\mathscr{T} - T_i$ is not a team cover of T for any $T_i \in \mathscr{T}$; and (2) for any team $T_i \in \mathscr{T}$ and $e_j \in T$, $(\mathscr{T} - T_i) \bigcup (T_i - e_j)$ is not a team cover. Given a base team T and its minimal team cover $\mathscr{T}$, the degree of an element $e_i \in T$ is the number of different teams in $\mathscr{T}$ that covers e_i through T, denoted by $\deg_i(T, \mathscr{T})$. The maximum degree of T and $\mathscr{T}$ is $\deg^{\max}(T, \mathscr{T}) = \max_{e_i \in T} \deg_i(T, \mathscr{T})$; the minimum degree of

T and $\mathscr{T}$ is $\deg^{\min}(T,\mathscr{T}) = \min_{e_i \in T} \deg_i(T,\mathscr{T})$. The *degree ratio* of T and $\mathscr{T}$ is $\mathrm{dr}(T,\mathscr{T}) = \frac{\deg^{\max}(T,\mathscr{T})}{\deg^{\min}(T,\mathscr{T})}$ and degree ratio of game $\mathscr{G}$ is $\mathrm{dr}(\mathscr{G}) = \max_{T,\mathscr{T}} \mathrm{dr}(T,\mathscr{T})$.

Theorem 6. *Assume we are given a fixed cost vector* $\mathbf{c}$*. We have* $\lambda \leq dr(\mathbf{c})$*. Here* $dr(\mathbf{c}) = \max_{\mathscr{T}} dr(T_{opt}(\mathbf{c}),\mathscr{T})$ *where* $\mathscr{T}$ *is a minimal team-cover of* $T_{opt}(\mathbf{c})$*.*

Proof. From the definition of $\mathbf{b}^{\min}$, for every $e_i \in T_{opt}(\mathbf{c})$, there exists a $T_i \in \mathscr{F}$ such that $e_i \notin T_i$ and $\sum_{e_j \in T_{opt}(\mathbf{c}) - T_i} x_j^{\min} = \sum_{e_j \in T_i - T_{opt}(\mathbf{c})} c_j$. Let $T_i' = T_i - T_{opt}(\mathbf{c})$ and $F_i = T_{opt}(\mathbf{c}) - T_i$. Then $\mathscr{T} = \{T_i' : e_i \in T_{opt}(\mathbf{c})\}$ is a team-cover of $T_{opt}(\mathbf{c})$. Let $\mathscr{T}^\star$ be any minimal team-cover that is a subset of $\mathscr{T}$. Without loss of generality, we assume $\mathscr{T}^\star = \{T_1', \ldots, T_k'\}$. Thus, $\mathrm{dr}(\mathbf{c}) \cdot \sum_{e_j \in T_{opt}(\mathbf{c})} x_j^{\min} \geq \sum_{i=1}^{k} \sum_{e_j \in F_i} x_j^{\min} = \sum_{i=1}^{k} \sum_{e_j \in T_i'} c_j$. On the other hand, T_i' covers F_i through T, $\sum_{e_j \in F_i} x_j^{\max} \leq \sum_{e_j \in T_i'} c_j$. Thus,

$$\sum_{e_j \in T_{opt}(\mathbf{c})} x_j^{\max} \leq \sum_{i=1}^{k} \sum_{e_j \in F_i} x_j^{\max} \leq \sum_{i=1}^{k} \sum_{e_j \in T_i'} c_j \leq \mathrm{dr}(\mathbf{c}) \cdot \sum_{e_j \in T_{opt}(\mathbf{c})} x_j^{\min}.$$

Therefore, $\lambda \leq \mathrm{dr}(\mathbf{c})$, which proves the theorem. □

Table 1. Summary of the frugalities for some binary demand games. Here, d is the maximum degree of a vertex in the given graph.

frugality ratio	VCG mechanism	Minimum frugality Truthful mechanism	Nash Implementation mechanism $\mathcal{M}^{out}$
Matroid Game	1	1	$1+\epsilon$
Unicast Game	$\Theta(n)$	$\Omega(\sqrt{n})$	$2+\epsilon$
Vertex Cover Game	$\Theta(d)$	$\Omega(\sqrt{d})$	$1+\epsilon$
Edge Cover Game	$\Theta(n)$	$\Omega(n)$	$d-1+\epsilon$

Next we show how to find the NE ratio via an example of the vertex cover game. In the vertex cover game, given a graph $G = (V,E)$ and each vertex v_i has a cost c_i, we need to find a subset $S \in V$ such that each edge has at least one vertex in S. Here every vertex is an element and a base-team is a minimum vertex cover. For the vertex cover game, we have the following lemma regarding $\mathrm{dr}(\mathbf{c})$.

Lemma 6. *For the vertex cover game, given a fixed cost vector* $\mathbf{c}$*, we have* $dr(\mathbf{c}) = 1$*.*

By definition, the NE ratio λ is at least 1. From Theorem 6 and Lemma 6, $\lambda \leq 1$. Thus, the NE ratio of a vertex set cover game is exactly one.

Theorem 7. *For the vertex cover game, the frugality of the Nash implementation mechanism computed by Algorithm 2 is* $1+\epsilon$ *and the frugality of the VCG mechanism is* $n-1$*, where* n *is the number of the vertices. For any truthful mechanism, the frugality is at least* $\Omega(\sqrt{n})$*.*

The frugality ratio of the edge cover game under various mechanisms is summarized in Theorem 8 below.

Theorem 8. *Given a graph G with maximum degree d, the frugality ratio of the Nash implementation mechanism computed by Algorithm 2 is $d-1+\epsilon$ and the frugality ratio of the VCG mechanism is $n-1$. Any truthful mechanism has the frugality ratio at least $\Omega(n)$*

Table 1 summarizes the frugality ratios of several binary demand games under three different mechanisms: the VCG mechanism, the minimum frugality ratio truthful mechanism and our Nash implementation mechanism $\mathcal{M}^{out}$ defined by Algorithm 2. We point out that sometimes the frugality ratio does not fully capture the over-payment of a mechanism. For example, even though the VCG mechanism has a comparable frugality ratio to Algorithm 2, the actual payment of the former can be $\Theta(n)$ times the latter. For example, consider the following spanning tree game. Given a cycle with only one edge having cost 1 while the others having cost 0. The VCG mechanisms pays $n-1$ while our mechanism only $1+\epsilon$.

6 Discussions

In this paper, we propose a class of mechanisms which implement social choice functions in Nash equilibria. Instead of relying on dominant strategy equilibria, these mechanisms aim at ensuring that any attainable Nash equilibrium will lead to the desirable outcome. We show that these mechanisms enjoy advantages over truthful mechanisms in terms of reduced payments and better frugality ratios.

Acknowledgment

We thank Professor Xiaotie Deng and his students for some helpful discussions.

References

1. Archer, A., Tardos, E.: Truthful mechanisms for one-parameter agents. In: Proceedings of the 42nd IEEE symposium on Foundations of Computer Science, pp. 482–491 (2001)
2. Archer, A., Tardos, E.: Frugal path mechanisms. In: Proceedings of the thirteenth annual ACM-SIAM symposium on Discrete algorithms, pp. 991–999 (2002)
3. Auletta, V., Prisco, R., Penna, P., Persiano, P.: Deterministic truthful approximation schemes for scheduling related machines. In: Proceedings of the 21st Symposium on Theoretical Aspects of Computer Science, Le Corum, Montpellier, France, pp. 608–619 (March 2004)
4. Clarke, E.H.: Multipart pricing of public goods. Public Choice, 17–33 (1971)
5. Groves, T.: Incentives in teams. Econometrica, 617–631 (1973)
6. Huang, C.-C., Kao, M.-Y., Li, X.-Y., Wang, W.: Using nash implementation to achieve better frugality ratios. Technical Report TR2007-604, Computer Science Department, Dartmouth College (2007)
7. Immorlica, N.: Computing With Strategic Agents. PhD thesis, MIT (2005)
8. Immorlica, N., Karger, D., Nikolova, E., Sami, R.: First-price path auctions. In: Proceedings of the 6th ACM Conference on Electronic Commerce, pp. 203–212 (2005)
9. Kao, M.-Y., Li, X.-Y., Wang, W.: Towards truthful mechanisms for binary demand games: A general framework. In: Proceedings of the 6th ACM conference on Electronic Commerce, pp. 213–222 (2005)

10. Karlin, A., Kempe, D., Tamir, T.: Beyond VCG: Frugality of truthful mechanisms. In: 46th Annual IEEE Symposium on Foundations of Computer Science, pp. 615–626 (2005)
11. Lehmann, D., O'callaghan, L.I., Shoham, Y.: Truth revelation in approximately efficient combinatorial auctions. Journal of ACM 49(5), 577–602 (2002)
12. Mu'alem, A., Nisan, N.: Truthful approximation mechanisms for restricted combinatorial auctions: extended abstract. In: Proceedings of the 18th National Conference on Artificial Intelligence, pp. 379–384 (2002)
13. Nisan, N., Ronen, A.: Algorithmic mechanism design. In: Proceedings of the 31st Annual Symposium on Theory of Computing, pp. 129–140 (1999)
14. Talwar, K.: The price of truth: Frugality in truthful mechanisms. In: Proceedings of the 20th Annual Symposium on Theoretical Aspects of Computer Science, pp. 608–619 (2003)
15. Vickrey, W.: Counterspeculation, auctions and competitive sealed tenders. Journal of Finance, 8–37 (1961)

The Price of Nash Equilibria in Multicast Transmissions Games

Vittorio Bilò

Dipartimento di Matematica "Ennio De Giorgi", Università del Salento
Provinciale Lecce-Arnesano P.O. Box 193, 73100, Lecce, Italy
vittorio.bilo@unile.it

Abstract. We consider the problem of sharing the cost of multicast transmissions in non-cooperative undirected networks with non-negative edge costs. In such a setting, there is a set of receivers R who want to be connected to a common source s. The set of choices available to each receiver $r \in R$ is represented by the set of all (s, r)-paths in the network. Given the set of choices performed by each receiver, a public known cost sharing method determines the cost share to be charged to each of them. Receivers are selfish agents aiming to receive the transmission at the minimum cost share and their interactions create a non-cooperative game. We study the problem of designing cost sharing methods yielding games whose price of anarchy (price of stability), defined as the worst-case (best-case) ratio between the cost of a Nash equilibrium and that of an optimal solution, is not too high. None of the methods currently known in the literature is able to achieve a good behavior on the price of anarchy and very little is known about their price of stability. We first give a lower bound on the price of stability of such methods, then we define and investigate some classes of cost sharing methods in order to characterize their weak points. Finally, we propose a new method, namely the *free-riders method*, which if from one hand it cannot improve in general on the price of anarchy of multicast transmission games, on the other one, it admits a polynomial time algorithm for computing a pure Nash equilibrium whose cost is at most twice the optimal one.

1 Introduction

In many real life situations we are faced with systems populated by independent selfish agents each operating according to their self-interest and optimizing an individual objective function. Selfish behavior naturally evokes the use of Game Theory [14] and Nash equilibria [12,13] in order to characterize solutions which can be consistent with the presence of non-cooperative agents.

With the affirmation of the Internet and huge networks in general, the study of non-cooperative networks has arisen as one of the most fervent research areas in Computer Science. Starting from the seminal paper by Koutsoupias and Papadimitriou [10], lots of traditional optimization problems on communication networks have been restated in a non-cooperative fashion and the notion of approximation ratio has been replaced by that of price of anarchy (the worst-case

T. Tokuyama (Ed.): ISAAC 2007, LNCS 4835, pp. 390–401, 2007.

ratio between the cost of a Nash equilibrium and that of an optimal solution) as a measure on how well a given algorithm or communication protocol can solve such problems. Network design, in particular, is an interesting area of application for this new approach. In such a setting, an alternative measure to the price of anarchy, that of price of stability [2], is often adopted. The price of stability measures the best-case ratio between the cost of a Nash equilibrium and that of an optimal solution. Its importance is due to the fact that a best Nash equilibrium can be considered as the optimal solution that can be proposed from which no player will defect.

In this paper, we consider the problem of sharing the cost of multicast transmissions in non-cooperative networks. In the classical version of the problem, we are given an undirected network $G = (V, E, c)$ with $c : E \mapsto \mathbf{R}^+$, a source node $s \in V$ and a set of ρ receivers $R \subseteq V \setminus \{s\}$. The objective is to find a minimum cost multicast tree, that is a tree of minimum total cost rooted at s and spanning the set of receivers R. Clearly, this is the problem of determining a minimum cost Steiner tree whose set of terminals is $R \cup \{s\}$. Now, assume that the cost of a multicast tree is shared among the receivers in order to recover exactly its cost. The algorithm used to compute the cost share to be charged to each receiver is called *cost sharing method*. In the non-cooperative version of this problem, each receiver is a selfish agent only interested in being charged the lowest possible cost share by the cost sharing method. Thus, a receiver may not agree on a particular solution (for instance, the minimum cost one) if she can be part of another solution yielding a lower cost share. Such a selfish behavior creates a non-cooperative game among ρ players (the receivers) which is assumed to end up to a certain pure Nash equilibrium, if any. The input network specifies the set of strategies available to the receivers: each receiver $r \in R$ can choose to adopt one of the possible (s, r)-paths in G. At each moment during the game the set of choices performed by all the receivers defines the state of the game. The state of the game is a subgraph of G containing an (s, r)-path for each $r \in R$, but it can no longer be a tree. The payoff achieved by each receiver on a particular state is given by the cost share charged to the receiver by the cost sharing method. Each time a receiver detects a better path, she performs an improving step (a move, for short). The sequence of moves performed by the receivers generates a sequence of transitions between different states. We say that a cost sharing method yields convergent games (or, that the method is convergent) if such a sequence of transitions is always guaranteed to end up at a Nash equilibrium starting from any initial state. The definition of a particular cost sharing method is the only instrument available to the network manager in order to keep the overall cost of the created network reasonably low. This is precisely the task we pursue in this paper: determining convergent cost sharing methods yielding a low price of anarchy and/or stability on all possible networks.

Related Works. The problem of sharing the cost of multicast transmissions in non-cooperative networks has been studied according to two main approaches. The first one is significantly different than ours and is based on the design of good cost sharing mechanisms rather than on good cost sharing methods. In the

underlying model, it is assumed that each player gets a certain utility from the transmission. Such a value is only known to the player. Each player is asked to report the network manager a utility value which may differ from the real one. According to the set of reported utilities, the cost sharing mechanism determines which players will receive the transmission, i.e., the set of receivers and the relative multicast tree, while the cost sharing method determines the cost share to be charged to each receiver. Since the reported utility values influence the cost shares, players may lie by misreporting their utilities in order to receive the transmission at a lower price. The main task pursued in this model is thus that of designing strategyproof cost sharing mechanisms, that is mechanisms for which the dominant strategy for each player is to declare her real utility value. This is usually obtained by combining a standard cost sharing mechanism due to Moulin and Shenker [11] together with one of two classical cost sharing methods, namely the marginal cost method and the Shapley value method [17]. Several papers [3,4,7,8,9,15,16] have addressed this problem.

The second approach is the one we adopt in this paper. As we have already explained, in this model the solution is obtained dynamically as an outcome of the non-cooperative game created by the particular cost sharing method adopted by the network manager. The problem of sharing the cost of multicast transmissions has been first considered in wireless networks [6] and then readdressed to traditional wired ones and extended with new results in [5]. In these papers four natural cost sharing methods are proposed and analyzed: the egalitarian, the path-proportional, the egalitarian-path-proportional and the Shapley value method. All methods but the path-proportional one are proved to yield convergent games and a high price of anarchy is shown for all of them. In particular, the egalitarian method has an unbounded price of anarchy, while all the others have price of anarchy equal to ρ. The more general problem of non-cooperative network design have been considered in [1]. In the underlying model, each player has a set of terminal nodes that she wants to connect and the price of stability of the Shapley value method is analyzed. From the results achieved in the paper, it follows that the Shapley value method has price of stability equal to H_ρ for directed networks. This results clearly yields the same upper bound on undirected networks (which is our scenario of investigation). The authors pose the determination of the price of stability of the Shapley value method in undirected networks as an interesting open problem. In the full version of their paper they show that, for the case of multicast transmissions as defined in our paper, it is equal to $4/3$ for the case $\rho = 2$ thus improving on the value $H_2 = 3/2$.

Our Contribution. We start our research by reconsidering the cost sharing methods presented in [5]. We encompass the path-proportional, the egalitarian-path-proportional and the Shapley value cost sharing methods into the more general class of weakly fair cost sharing methods (see Section 2) and give the first lower bounds on their price of stability. We note that the instances yielding the worst-case lower bound on the price of anarchy of all known methods are usually of the same form. We can also prove formally this empirical result for

a subclass of cost sharing methods that we call topology independent strongly fair monotone methods.

The intuition behind the analysis of these situations suggested us a new weakly fair method, which we call the free-riders method. We prove that this method yields convergent games and design a polynomial time algorithm for computing one of its pure Nash equilibria. If from one hand the price of anarchy of the free-riders method remains ρ on some extreme situations, we prove that the cost of the Nash equilibrium computed by our algorithm is at most twice that of an optimal solution and present also an instance showing that the analysis is tight. This gives also an upper bound on the price of stability of the free-riders method for which we also present a lower bound equal to $\frac{\rho+2}{\rho+1}$.

As suggested by its name, the free-riders method encourages the phenomenon of free-riders and so it can be considered somehow unfair. However, we prove that any weakly fair method allowing no free-riders cannot achieve a price of anarchy equal to 1. Moreover, no weakly fair method which is topology independent (that is, a method always computing the same cost shares on the same states no matter whether the underlying network topology may be different) can achieve a price of stability equal to 1.

We list the results of this paper, together with the ones already known in the literature, in Figure 1.

Cost sharing method	Price of stability	Price of anarchy
Egalitarian	1	∞ [5]
Egalitarian-path-proportional	$[\frac{2\rho}{\rho+1}; \rho]$ [$*$]	ρ [5]
Path-proportional	$[\frac{1.915\rho}{\rho+1.313}; \rho]$ [$*$]	ρ [5]
Shapley value	$[\frac{12\rho}{7\rho+5}; H_\rho]$ [1][$*$]	ρ [5]
Free-riders method	$[\frac{\rho+2}{\rho+1}; 2]$ [$*$]	ρ [$*$]

Fig. 1. Results for the price of Nash equilibria on multicast transmission games. Label [$*$] denotes the achievements of this paper.

Paper Organization. The paper is organized as follows. Next section contains the necessary definitions and notation. Section 3 addresses the class of fair cost sharing methods, while in Section 4 we present and analyze the free-riders method. Finally, in the last section we give some concluding remarks and discuss open problems. Due to space limitations all proofs have been removed and will be given in the full version of the paper.

2 Definitions and Notation

An instance $I = (G, R, s)$ of the multicast transmissions game is defined by an undirected network $G = (V, E, c)$ with $c : E \mapsto \mathbf{R}^+$, a source node $s \in V$ and a set of ρ receivers $R \subseteq V \setminus \{s\}$. Each receiver $r \in R$ has a set of strategies

$\mathcal{P}_r(G,s)$ which consists of all the (s,r)-paths in G. We denote as π_r the strategy chosen by receiver r. The union $\Pi = \bigcup_{r\in R}\pi_r$ of the strategies played by all the receivers is a state of the game and we define $\Pi(I)$ as the set of all possible states which can be obtained on instance I. The cost of a state Π is defined as $c(\Pi) = \sum_{e\in\Pi} c(e)$, while $c(\pi_r) = \sum_{e\in\pi_r} c(e)$ is the cost of the (s,r)-path in Π. Once fixed a particular cost sharing method $\mathcal{M}$, we consider the non-cooperative game induced by $\mathcal{M}$ on I.

Definition 1. *A cost sharing method $\mathcal{M}$ is a function that, given an instance $I = (G,R,s)$ and a state Π, associates a cost share $\mathcal{M}_r(I,\Pi) \geq 0$ to each receiver $r \in R$ in such a way that $\sum_{r\in R}\mathcal{M}_r(I,\Pi) = c(\Pi)$.*

The payoff achieved by player r on state Π is defined as the cost share $\mathcal{M}_r(I,\Pi)$ computed by the cost sharing method $\mathcal{M}$. We denote as T^* a minimum cost Steiner tree connecting the required set $R\cup\{s\}$ and as $\mathcal{T}(\mathcal{M},\mathcal{I})$ the set of pure Nash equilibria for the multicast transmission game induced by $\mathcal{M}$ on instance I. The price of anarchy (resp. stability) of $\mathcal{M}$ on instance I is defined as $PoA(\mathcal{M},I) = \max_{\Pi\in\mathcal{T}(\mathcal{M},\mathcal{I})}\frac{c(\Pi)}{c(T^*)}$ (resp. $PoS(\mathcal{M},I) = \min_{\Pi\in\mathcal{T}(\mathcal{M},\mathcal{I})}\frac{c(\Pi)}{c(T^*)}$).

We are interested in determining cost sharing methods inducing games whose prices of anarchy and stability are as low as possible on any possible instance. Hence we define the price of anarchy (resp. stability) of $\mathcal{M}$ as $PoA(\mathcal{M}) = \sup_I PoA(\mathcal{M},I)$ (resp. $PoS(\mathcal{M}) = \sup_I PoS(\mathcal{M},I)$).

As it can be easily imagined, one may design plenty of different cost sharing methods. In the sequel we outline some properties allowing us to classify such methods into general classes.

Definition 2. *A cost sharing method $\mathcal{M}$ is*

- weakly fair *if $\mathcal{M}_r(I,\Pi) \leq c(\pi_r)$;*
- strongly fair *if the cost of each edge $e\in\Pi$ is only shared among its users, hence, if $\mathcal{M}_r(I,\Pi,e)$ denotes the fraction of $c(e)$ charged by $\mathcal{M}$ to r, we have $\mathcal{M}_r(I,\Pi) = \sum_{e\in\pi_r}\mathcal{M}_r(I,\Pi,e)$;*
- pure *if $\mathcal{M}_r(I,\Pi) > 0$ for each $r\in R$ such that $c(\pi_r) > 0$;*
- aggregating *if each of its Nash equilibria is a tree;*
- topology independent *if for any two instances $I=(G,R,s)$ and $I'=(G',R,s)$ and any state $\Pi\in\Pi(I)\cap\Pi(I')$ it holds $\mathcal{M}_r(I,\Pi) = \mathcal{M}_r(I',\Pi)$ for each $r\in R$.*

Clearly, it follows from definition that a strongly fair method is also weakly fair since $\mathcal{M}_r(I,\Pi) = \sum_{e\in\pi_r}\mathcal{M}_r(I,\Pi,e) \leq \sum_{e\in\pi_r} c(e) = c(\pi_r)$.

The majority of natural cost sharing methods one can think of are fair ones. This seems perfectly reasonable: why should a receiver pay for resources she is not using? However, this is really true only for strongly fair methods. Weakly fair methods, in fact, only give the receiver the feeling of paying only for the resources she is using, since the cost share can also be influenced by the cost of the resources used exclusively by other receivers (see for example the egalitarian-path-proportional method). On the other hand, also pure methods are highly

desired: why should a receiver not pay for the resources she is using? The definition of aggregating methods, instead, is more related to the notion of efficiency of Nash equilibria: if a Nash equilibrium is not a tree, hence containing cycles, it is more likely to yield a high price of anarchy. Finally, topology independent methods guarantee that the cost share is only a function of the current state and, not requiring a possibly onerous analysis of the network topology, are easier to be computed. Throughout the paper, we will only deal with topology independent cost sharing methods, hence, in order to simplify the notation, we will write $\mathcal{M}_r(\Pi)$ instead of $\mathcal{M}_r(I, \Pi)$ when the instance $I = (G, R, s)$ is fixed.

Given a state Π, let $n_e(\Pi)$ be the number of receivers using edge e in their strategies. We introduce an ordering on the $n_e(\Pi)$ receivers using e and use $\sigma_e(\Pi, r)$ to define the position occupied by receiver r in the ordering defined on edge e at state Π. A natural way to define such an ordering is to consider the evolutionary behavior of the receivers during the game. One may start with an arbitrary ordering defined on the initial state of the game. Then, each time a receiver r changes her strategy from π_r to π'_r, thus letting the game evolving from state Π to state Π', we have $\sigma_e(\Pi', r') = \sigma_e(\Pi, r') - 1$ for each $e \in \pi_r \setminus \pi'_r$ and for each receiver r' such that $\sigma_e(\Pi, r') > \sigma_e(\Pi, r)$, while we have $\sigma_e(\Pi', r) = n_e(\Pi')$ for each $e \in \pi'_r \setminus \pi_r$.

Definition 3. *A cost sharing method $\mathcal{M}$ is* dynamic *if $\mathcal{M}_r(I, \Pi)$ is a function of $\sigma_e(\Pi, r)$.*

In such a setting, a state of the game is thus represented by the pair (Π, σ).

Throughout the paper we will make an intensive use of the parallel link graph in order to model worst-case behavior of several cost sharing methods. The use of such a topology in our multicast transmission games may look like a nonsense since the parallel link graph is a multigraph and it assumes the presence of multiple receivers residing at the same node and both of these properties are indeed not allowed in our model. However, we stress here that the parallel link graph can be seen as a compact way to represent a particular legal network topology for our games as shown in Figure 2.

All the receivers have two different strategies in the first network, call it G, that is, the one using the edge of cost x and the one using the edge of cost y.

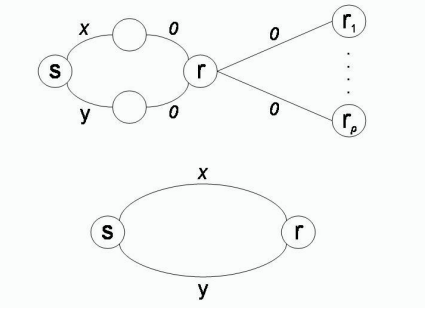


Fig. 2. A network and its compact representation as a parallel link graph

The same set of strategies, together with their cost, persists in the parallel link graph representation G'. Hence, we have a bijection B between the set of states of both the two networks. Clearly, for each considered cost sharing method and receiver r it must be proved that $\mathcal{M}_r(G, R, s, \Pi) = \mathcal{M}_r(G', R, s, B(\Pi))$.

The four cost sharing methods, proposed and studied so far in the literature are:

- the egalitarian method [5], defined as $\mathcal{M}_r(I, \Pi) = \frac{c(\Pi)}{\rho}$;
- the egalitarian-path-proportional method [5], defined as $\mathcal{M}_r(I, \Pi) = \frac{c(\Pi) \cdot c(\pi_r)}{\sum_{r \in R} c(\pi_r)}$;
- the path-proportional method [5], defined as $\mathcal{M}_r(I, \Pi, e) = \frac{c(e) \cdot c(\pi_r)}{\sum_{r: e \in \pi_r} c(\pi_r)}$;
- the Shapley value method [1,5], defined as $\mathcal{M}_r(I, \Pi, e) = \frac{c(e)}{n_e(\Pi)}$.

The fact that the last three methods are pure and topology independent is quite clear from their definition. The path-proportional method and the Shapley value method are also strongly fair. For the egalitarian-path-proportional one we prove the following result.

Proposition 1. *The egalitarian-path-proportional method is weakly fair.*

As to the aggregating property, the egalitarian method is clearly an aggregating one since removing cycles makes the overall cost of the state decrease. Also the Shapley value method is aggregating as shown in [6]. We now prove that the remaining two methods do not satisfy this property.

Proposition 2. *The egalitarian-path-proportional and the path-proportional methods are not aggregating.*

3 Fair Cost Sharing Methods

The only non-fair method proposed in the literature is the egalitarian one which, if from one hand is the only known method achieving a price of stability equal to 1, from the other hand has a price of anarchy infinitively high.

The first result we propose in this section is an upper bound on the price of anarchy of weakly fair cost sharing methods. This follows by extending the proof presented in [5] for the path-proportional, egalitarian-path-proportional and Shapley value cost sharing methods.

Theorem 1. *$PoA(\mathcal{M}) \leq \rho$ for any weakly fair cost sharing method $\mathcal{M}$.*

In order to obtain a matching lower bound on the price of anarchy of a fair cost sharing method, we prove the following interesting result for the subcase of topology independent strongly fair monotone methods. To this aim, we say that a strongly fair method is monotone when adding a new user to an edge e strictly decreases the maximum cost share yielded by e. More formally, let $\mathcal{M}$ be a strongly fair method. We say that $\mathcal{M}$ is monotone if for each edge e and

for each pair of states Π' and Π'' such that $|R'_e| = |\{r \in R : e \in \pi'_r\}| < |R''_e| = |\{r \in R : e \in \pi''_r\}|$, it holds $\max_{r \in R'_e} \mathcal{M}_r(\Pi', e) > \max_{r \in R''_e} \mathcal{M}_r(\Pi'', e)$.

Theorem 2. *If the price of anarchy of a topology independent strongly fair monotone cost sharing method $\mathcal{M}$ is ρ, then there exists (an inconsequential generalization of) a parallel link graph G' for which $PoA(G', R, s, \mathcal{M}) = \rho$.*

Besides of being topology independent strongly fair, the path-proportional and the Shapley value cost sharing methods are also monotone. Hence, by applying the above theorem we can simplify the proofs presented in [5] for the lower bounds on the price of anarchy yielded by these methods as it can be easily seen on the parallel link graph depicted in Figure 2 where we set $x = 1$ and $y = \rho$. Interestingly enough, such a worst-case behavior applies also to the egalitarian method by setting $y = \epsilon$, with $\epsilon > 0$ arbitrarily small, so as to obtain an unbounded price of anarchy. The only exception is the egalitarian-path proportional method which, in fact, is neither strongly fair nor monotone. As shown in [5], a price of anarchy of ρ can be forced on this method by using a generalization of the parallel link graph. Thus, even though it is rather difficult to prove this claim in general, since one may design plenty of fancy cost sharing methods, one can conjecture that the parallel link graph and its generalizations are likely to exhibit the worst-case behavior on the price of anarchy of each of the methods we have in mind.

As to the price of stability of the methods considered so far, no results are known in the literature except for the trivial upper bound of H_ρ for the Shapley value method coming from the results presented in [1] for directed networks and that the egalitarian method has price of stability equal to 1. We provide a first characterization of the lower bounds on the price of stability of these methods in the following theorem.

Theorem 3. *The price of stability of the Shapley value cost sharing method is at least $\frac{12\rho}{7\rho+5}$ if ρ is odd and at least $\frac{12\rho}{7\rho+4}$ if ρ is even, that of the path-proportional method is at least $\frac{1.588\rho}{0.829\rho+1.088}$, while for any $\rho \geq 3$ that of the egalitarian-path-proportional method is at least $\frac{2\rho}{\rho+1}$.*

We stress that for $\rho = 2$ our lower bound for the Shapley value method matches the one given in [1] which is known to be tight.

4 The Free-Riders Cost Sharing Method

As an informal hint provided by Theorem 2 and its consequent discussion, we have that, if we wish to design a cost sharing method yielding a low price of anarchy, we should probably think of a method having good performances on parallel link graphs.

Starting from this intuition, we propose a topology independent strongly fair dynamic cost sharing method that we call the *free-riders method*.

Definition 4 (Free-riders method)

$$\mathcal{M}_r(I, \Pi, \sigma, e) = \begin{cases} c(e) & \text{if } \sigma_e(\Pi, r) = n_e(\Pi), \\ 0 & \text{otherwise.} \end{cases}$$

According to this method, each edge e is paid for by one and only one receiver, hence all other receivers using e act as free-riders. It is easy to see that on the instances yielded by parallel link graphs as depicted in Figure 2, the price of anarchy of the free-riders method is 1 as long as $x \neq y$. In the case in which $x = y$, we have that each solution in which some of the receivers use the link of cost x and some other ones use that of cost y is a Nash equilibrium for the free-riders method. This shows that such a method is not aggregating. By generalizing this situation to the case in which there are ρ parallel links all having the same cost and considering the Nash equilibrium in which each receiver uses a different link, a lower bound of ρ on the price of anarchy of the free-riders method can be still obtained. An upper bound of ρ for the price of anarchy comes from Theorem 1, since the free-riders method is weakly fair.

In order to be proposed in practice, we have to prove that this new method yields convergent games. This is done in the following theorem.

Theorem 4. *The free-riders method yields convergent games.*

An interesting property is that each minimum cost path tree yields a Nash equilibrium for the free-riders method.

Lemma 1. *Each state (Π, σ) such that Π is a minimum cost path tree for the set of receivers R in G is a Nash equilibrium for the free-riders method.*

This lemma suggests us directly a suitable instance for lower bounding the price of anarchy of the free-riders method even when restricting to Nash equilibria exhibiting the tree property.

Theorem 5. *The price of anarchy of the free-riders method is ρ even when restricting to Nash equilibria which are trees.*

Lemma 1 gives us a polynomial time algorithm for computing a Nash equilibrium yielded by the free-riders method. However, we have seen that such an equilibrium may have a cost much greater than the minimum cost multicast tree. In the sequel we show how to design a polynomial time algorithm computing Nash equilibria with significantly better performances.

Given a path π we denote as $V(\pi)$ the set of nodes touched by π. Given a set of nodes A and a receiver $r_i \notin A$, we define the distance of r_i from A as the minimum among the distances between r_i and any other node belonging to A. Let $v(i)$ be the node attaining such a minimum, we call the $(v(i), r_i)$-path in G, the connecting path of r_i.

Algorithm NASH EQUILIBRIUM which, given an instance (G, R, s), computes a Nash equilibrium for the game yielded by the free-riders method is given below.

Algorithm NASH EQUILIBRIUM:
Input: an instance (G, R, s).
Output: a Nash equilibrium (Π, σ) for the game yielded by the free-riders method on (G, R, s).
1. $\Pi = \emptyset$
2. $A = \{s\}$
3. while $R \cap A \neq R$ do
4. let r_i be the receiver at minimum distance from A and let π_i be her connecting path
5. $\Pi = \Pi \cup \pi_i$
6. $A = A \cup V(\pi_i)$
7. for each $e \in \Pi$ set $\sigma_e(\Pi, r_i) = n_e(\Pi)$ if $e \in \pi_i$
8. output(Π, σ)

We first show that the computed solution is a Nash equilibrium.

Theorem 6. *The solution (Π, σ) computed by NASH EQUILIBRIUM is a Nash equilibrium for the free-riders method.*

In order to bound the ratio between the cost of the Nash equilibrium (Π, σ) computed by our algorithm and that of an optimal solution, it suffices noting that $c(\Pi)$ is at most the cost of a minimum spanning tree rooted at s for the set of receivers R. Such a value is known to be at most twice that of a minimum Steiner tree with terminal vertices given by R. Hence, we can claim the following theorem.

Theorem 7. *Let (Π, σ) be a Nash equilibrium computed by our algorithm. It holds $\frac{c(\Pi)}{c(T^*)} \leq 2$.*

By executing our algorithm on the network depicted in Figure 3 with $x = 2$, $y = 1$ and $z = 1 + \epsilon$, it is possible to see that such a bound is almost tight. From this result we obtain the following corollary about the upper bound on the price of stability of the free-riders method.

Corollary 1. *The price of stability of the free-riders method is at most 2.*

As for the lower bound, we can prove that it is strictly greater than 1.

Theorem 8. *The price of stability of the free-riders method is at least $\frac{\rho+2}{\rho+1}$.*

The free-riders method is thus the first cost sharing method for which we have a polynomial time algorithm for computing one of its equilibria and for which we can prove a constant upper bound on the price of stability. Clearly, it has

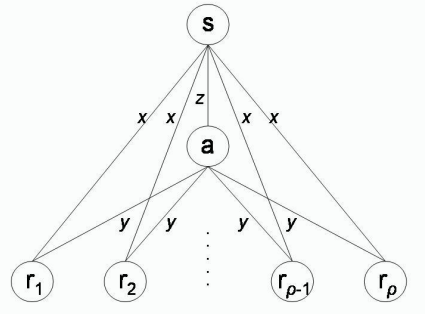


Fig. 3. A network

the drawback of allowing the proliferation of free-riders in the network, thus in a sense it can be considered an unfair method. In the next result, however, we show that no pure weakly fair method can achieve a price of anarchy equal to 1. We will prove, indeed, a stronger result for which we need a more general definition of pure methods. To this aim, we say that a weakly fair cost sharing method $\mathcal{M}$ is ϵ-pure if each receiver pays for at least ϵ times the cost of the path she uses, i.e., $\mathcal{M}_r(\Pi) \geq \epsilon \sum_{e \in \pi_r} c(e)$ for each $r \in R$ and each state Π. Since $\mathcal{M}_r(\Pi) \geq 0$ for each $r \in R$ and each state Π by definition, and there may exist graphs for which a path can be shared by all the receivers, reasonable values for such an ϵ will fall in the interval $(0, \frac{1}{\rho}]$.

Theorem 9. *No ϵ-pure weakly fair method can achieve a price of anarchy smaller than $\frac{1}{1-\epsilon(\rho-1)}$, for any $\epsilon \in (0; \frac{1}{\rho}]$.*

As another impossibility result, we show that a topology independent weakly fair cost sharing method cannot achieve a price of stability equal to 1.

Theorem 10. *Any topology independent weakly fair cost sharing method cannot have a price of stability equal to 1.*

5 Conclusions

We have studied the problem of sharing the cost of multicast transmissions in non-cooperative networks and analyzed the price of Nash equilibria of some cost sharing methods. The main achievement of this paper is the design and analysis of a new method, the free-riders method. All the methods proposed in the literature have a poor behavior with respect to their price of anarchy which may either be unbounded, or grow linearly with the number of receivers. The free-riders method does not improve directly on this bound, but gives us a polynomial time algorithm computing a Nash equilibrium whose cost is at most twice that of an optimal solution, thus making the free-riders method of practical use. Another interesting contribution of this paper is the effort of classifying the cost sharing methods according to some common properties and trying to outline powers and limits of the various classes. It seems that, in general, unfair methods can yield better performances with respect to fair ones; it would be intriguing to explore deeply this situation. This is only an initial work and several improvements and open problems are issued by our research.

The most important one is, of course, the determination of an upper bound on the price of stability of the other cost sharing methods considered in the paper. The classical techniques for upper bounding the price of stability, based on the exploitation of a potential function, cannot be applied in our case. The only exception is represented by the Shapley value method whose upper bound, however, is significant only for the directed case. It would also be interesting to understand if there is a correlation between the price of anarchy and that of stability yielded by each particular instance. For example, all instances yielding a price of anarchy equal to ρ for the considered cost sharing methods yield a price of stability equal to 1. Is this relationship only casual?

Finally, it would be intriguing to consider the effects of combining two or more methods. For example one can share half of the cost of each edge according to the free-riders method and the other half according to the Shapley value method.

References

1. Anshelevich, E., Dasgupta, A., Kleinberg, J., Tardos, E., Wexler, T., Roughgarden, T.: The Price of Stability for Network Design with Fair Cost Allocation. In: Proceedings of the 45th Annual IEEE Symposium on Foundations of Computer Science (FOCS), pp. 295–304. IEEE Computer Society, Los Alamitos (2004)
2. Anshelevich, E., Dasgupta, A., Tardos, E., Wexler, T.: Near-Optimal Network Design with Selfish Agents. In: Proceedings of the 35th Annual ACM Symposium on Theory of Computing (STOC), pp. 511–520. ACM Press, New York (2003)
3. Archer, A., Feigenbaum, J., Krishnamurthy, A., Sami, R., Shenker, S.: Approximation and collusion in multicast cost sharing. Games and Economic Behavior 47(1), 36–71 (2004)
4. Bilò, V., Di Francescomarino, C., Flammini, M., Melideo, G., Moscardelli, L., Navarra, A.: Sharing the cost of multicast transmissions in wireless networks. Theoretical Computer Science (to appear)
5. Bilò, V., Fanelli, A., Flammini, M., Melideo, G., Moscardelli, L.: Multicast transmissions in non-cooperative networks with a limited number of selfish moves (submitted)
6. Bilò, V., Flammini, M., Melideo, G., Moscardelli, L.: On Nash Equilibria for Multicast Transmissions in Ad-Hoc Wireless Networks. Wireless Networks (to appear)
7. Feigenbaum, J., Krishnamurthy, A., Sami, R., Shenker, S.: Hardness results for multicast cost sharing. Journal of Public Economics 304(1-3), 215–236 (2003)
8. Feigenbaum, J., Papadimitriou, C., Shenker, S.: Sharing the cost of multicast transmissions. In: Proceedings of 32nd ACM Symposium on Theory of Computing (STOC), pp. 218–227. ACM Press, New York (2000)
9. Jain, K., Vazirani, V.V.: Applications of approximation algorithms to cooperative games. In: Proceedings of 33rd ACM Symposium on Theory of Computing (STOC), pp. 364–372. ACM Press, New York (2001)
10. Koutsoupias, E., Papadimitriou, C.: Worst-case equilibria. In: Meinel, C., Tison, S. (eds.) STACS 1999. LNCS, vol. 1563, pp. 404–413. Springer, Heidelberg (1999)
11. Moulin, H., Shenker, S.: Strategyproof sharing of submodular costs: budget balance versus efficiency. Economic Theory 18(3), 511–533 (2001)
12. Nash, J.: Equilibrium points in n-person games. Proceedings of the National Academy of Sciences 36, 48–49 (1950)
13. Nash, J.: Non-cooperative games. Annals of Mathematics 54(2), 286–295 (1951)
14. Osborne, M.J., Rubinstein, A.: A course in Game Theory. MIT Press, Cambridge (1994)
15. Penna, P., Ventre, C.: More powerful and simpler cost-sharing methods. In: Persiano, G., Solis-Oba, R. (eds.) WAOA 2004. LNCS, vol. 3351, p. 97. Springer, Heidelberg (2005)
16. Penna, P., Ventre, C.: Free-riders in steiner tree cost-sharing games. In: Pelc, A., Raynal, M. (eds.) SIROCCO 2005. LNCS, vol. 3499, pp. 231–245. Springer, Heidelberg (2005)
17. Shapley, L.S.: The value of n-person games. Contributions to the theory of games, pp. 31–40. Princeton University Press (1953)

An Efficient Algorithm for Enumerating Pseudo Cliques

Takeaki Uno

National Institute of Informatics
2-1-2, Hitotsubashi, Chiyoda-ku, Tokyo 101-8430, Japan
uno@nii.jp

Abstract. The problem of finding dense structures in a given graph is quite basic in informatics including data mining and data engineering. Clique is a popular model to represent dense structures, and widely used because of its simplicity and ease in handling. Pseudo cliques are natural extension of cliques which are subgraphs obtained by removing small number of edges from cliques. We here define a pseudo clique by a subgraph such that the ratio of the number of its edges compared to that of the clique with the same number of vertices is no less than a given threshold value. In this paper, we address the problem of enumerating all pseudo cliques for given a graph and a threshold value. We first show that it seems to be difficult to obtain polynomial time algorithms using straightforward divide and conquer approaches. Then, we propose a polynomial time, polynomial delay in precise, algorithm based on reverse search. We show the efficiency of our algorithm in practice by computational experiments.

1 Introduction

Let us consider the problem of finding dense structures from a given graph, i.e., finding vertex induced subgraphs including many edges. In graphs representing similarity or relation, dense structures can be considered to represent groups of similar objects or deeply related objects. Thus, the problem is important in many scientific areas. This is especially true in data engineering and data mining, where it is one of basic problems, and has many applications such as clustering, community discovering, machine learning, Web search, etc. [1,6,8,9,10,11,14,20]. A clique is a subgraph that is a complete graph. It is a fundamental structure for representing dense structures. It has several good mathematical properties, and can be easily handled. As a result, clique detection has been used in many researches. Cliques are considered as a part of dense structures, or seeds of dense structures. The clique enumeration can be efficiently carried out thanks to the increase of computation power, and new algorithms[12,16]. Currently, the bottle neck of the computation in the practice is usually the output process, thus the algorithm is almost optimal.

Then, as a next step, people wanted to use a richer model than cliques. In very sparse graphs, a subgraph containing only small cliques can be considered

T. Tokuyama (Ed.): ISAAC 2007, LNCS 4835, pp. 402–414, 2007.

as a dense structure if it has many edges when compared to the others. If the data is incorrect so that some edges are missing, then a vertex set should be a clique will not be a clique. For robust computation, "pseudo cliques" should be used. For example, such pseudo cliques are used for web page clustering[9].

We can consider several models to represent pseudo cliques. A possible model is a subgraph that is obtained from a clique by removing at most θ edges, where θ is a given threshold constant number. An advantage of this model is that in this definition any subset of a pseudo clique is also a pseudo clique in the sense of the vertex subset, thus the family of pseudo cliques satisfies the monotone property. This is a useful property, and we can use many techniques in the literatures to develop an efficient algorithm. However, a disadvantage of the model is that for graphs of any size the threshold is the same, thus larger subgraphs can lose only a small portion of its edges. This is contrary to the intuitions. Moreover, so many small vertex sets will become pseudo cliques.

Another model is to define a pseudo clique by a subgraph that has at least a constant ratio of edges compared to a clique of the same size. Precisely, we define the *density* of a subgraphs by the number of edges over the number of all its vertex pairs. A subgraph is a pseudo clique if its density is no less than the given threshold value. In this definition, the family of pseudo cliques no longer satisfies the monotone property. It is a disadvantage of this definition. On the other hand, small subgraphs are pseudo cliques only if they are cliques, since the limitation of the number of edge removals changes as the size of subgraphs. This is an advantage of this definition.

In the literatures, a subgraph having many edges compared to the number of its vertices is often called *dense subgraph*, *heavy graph*, or *maximum edge subgraph*. However, all these terms are used to represent the subgraph having the highest ratio of edges, thus it is mainly used in optimization. On the other hand, in many areas in informatics, a subgraph having many edges thereby similar to clique is often called a "pseudo clique" or "quasi clique". In this paper, we use the term pseudo clique.

The problem of maximizing the density among subgraphs of given size k is NP-complete since it includes the maximum clique problem as its special case[4]. However, if $k = \Theta(|V|)$ holds, there is a PTAS algorithm[2]. For the edge weighted version, an $O(|V|^{1/3-\epsilon})$ approximation algorithm is known[4]. However, finding an induced subgraph maximizing average degree in it can be solved in polynomial time[4].

In this paper, we address the enumeration problem of all pseudo cliques in a given graph. We choose the latter definition for pseudo cliques. We first show the existence of polynomial delay algorithm is non-trivial, by proving that straightforward back-tracking (branch-and-bound) approaches involve an NP-complete problem in each iteration. Note that it does not assure that non-existence of output polynomial time algorithm even when $P \neq NP$.

In contrast, a reverse search works well for this problem. We will show that for any pseudo clique, one of its vertex satisfies that its removal is also a pseudo clique. From this, we can obtain an adjacency relation on pseudo cliques spanning

all the pseudo cliques. Then we define a tree-shaped traversal route on all pseudo cliques by this adjacency. Our algorithm traverses the route in a depth-first manner with taking polynomial time on each pseudo clique, thus it is polynomial delay algorithm. We then introduce a method to decrease the time complexity and practical computation time. By computational experiments we show the efficiency of our algorithm in practice.

The organization of this paper is as follows. Section 2 is for preliminaries. In Section 3, we present the hardness result for straightforward approach. Section 4 describes the adjacency and the algorithm. In Section 5, we present the results of our computational experiments, and conclude the paper in Section 6.

2 Preliminary

Let $G = (V, E)$ be a graph with a vertex set V and an edge set $E \subseteq V \times V$. In this paper we consider graphs with no multiple edge. We say vertex v is *adjacent* to vertex u if there is an edge $e = \{u, v\}$ in E. We denote the degree of v by $deg(v)$, and the maximum degree by Δ.

For a vertex set $U \subseteq V$, the *induced subgraph* by U, denoted by $G[U] = (U, E[U])$, is a graph composed of edges of G whose endpoints are both in U. $G[U]$ is also called an *induced subgraph*. In Fig. 1, the subgraph induced by vertex set $\{3, 5, 6, 9, 11\}$ is the graph inside the gray circle without edges one of its endpoint outside the circle. If $U = \emptyset$, we define $G[U]$ by the empty graph $(\emptyset, \emptyset)$. If any two vertices in U are connected by an edge, U and $G[U]$ are called a *clique*. In Fig 1, vertices $\{5, 7, 9, 11\}$ form a clique. For a vertex v and a vertex set U, we denote the number of edges connecting v and a vertex in U by $deg_U(v)$.

Let $clq(n) = \frac{n(n-1)}{2}$, where $clq(n)$ is the number of edges in the clique of n vertices. For a vertex subset $U \subseteq V$ at a size of at least 2, the *density* of $G[U]$ is defined by $|E[U]|/clq(|U|)$. The density is the ratio of the edges in $G[U]$ compared to the complete graph of $|U|$ vertices. In Fig 1, the density of the subgraph induced by $K = \{3, 5, 6, 9, 11\}$ is 7/10. We define the density of $G[\emptyset]$ and graphs with one vertex by 1. Suppose that $\theta, 0 \leq \theta \leq 1$ be a constant number called *threshold*. Then, an induced subgraph $G[U]$ is called a *pseudo clique* if its density is no less than θ. We note that $G[\emptyset]$ and $G[\{v\}]$ for any vertex v are pseudo cliques for any threshold.

Let w be an edge weight function $w : E \to R$. For an edge subset $F \subseteq E$, we define the weight of F by the sum of weights of edges in F, and denote it by $w(F)$. We also define the weight of $G[U]$ by $w(E[U])$. For a given edge weight function $w : E \to R$, we define the *weighted density* of an induced subgraph $G[U]$ by $w(E[U])/clq(|U|)$. We define the weighted density of the graphs with at most one vertex by $+\infty$. For a threshold $\theta \in R$, an induced subgraph $G[U]$ is a *weighted pseudo clique* if its weighted density is no less than θ.

We define the (weighted) density of a vertex set by that of the subgraph induced by the vertex set. We often say U is a (weighted) pseudo clique if $G[U]$ is a (weighted) pseudo clique. We here address the following enumeration problem.

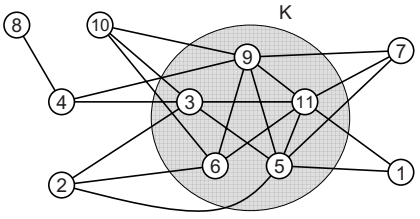


Fig. 1. An example of pseudo clique and other vertices

(Weighted) Pseudo Clique Enumeration Problem
For given graph $G = (V, E)$ and threshold θ, (edge weight function w), output all vertex sets whose induced subgraphs are (weighted) pseudo cliques of G

If an enumeration algorithm which outputs a set of solutions terminates in time polynomial of the sum of the input and output sizes, the algorithm is said to be *output polynomial time*. The longest computation time between the output of any two consecutive solutions is called the *delay*. An algorithm is *polynomial delay* if the delay is polynomial in the input size. The computation time of a polynomial delay algorithm is linear in the output size, thus it is optimal for the output size and considered to be practically efficient. Our goal in this paper is to develop a polynomial delay algorithm for pseudo clique enumeration problem.

3 Hardness Result for Straightforward Approaches

A popular scheme for constructing an enumeration algorithm is so called binary partition, which is a variant of branch and bound method. For the problem of enumerating all the elements of an implicitly given subset family $\mathcal{F} \subseteq 2^E$, binary partition algorithm works in the following way. If $|\mathcal{F}|$ is less than a certain constant number, it enumerates elements in $\mathcal{F}$ directly. Otherwise, it chooses an element $e \in E$ and divide $\mathcal{F}$ into two sets $\mathcal{F}^+$ and $\mathcal{F}^-$ so that $\mathcal{F}^+$ consists of all the elements of $\mathcal{F}$ including e, and $\mathcal{F}^-$ consists of those not including e. Then, we check whether each of $\mathcal{F}^+$ and $\mathcal{F}^-$ is the emptyset or not, and if it is not empty, we enumerate the elements recursively, by dividing $\mathcal{F}^+$ or $\mathcal{F}^-$ by choosing another element e'.

The algorithms so called "depth first search" or "back tracking" are also considered to be a variant of binary partition algorithms. Binary partition works for many problems, such as spanning trees, paths, and cycles[15], cliques and independent sets[12], perfect matchings in bipartite graphs[5], etc.

The number of iterations of a binary partition algorithm is linear in $|\mathcal{F}|$, which is the size of output, thanks to the check whether either $\mathcal{F}^+$ or $\mathcal{F}^-$ is an emptyset. Thus, if the check can be done in polynomial time of the input size, the binary partition algorithm is polynomial delay. When we want to enumerate the pseudo cliques by binary partition, the check problem is defined as follows.

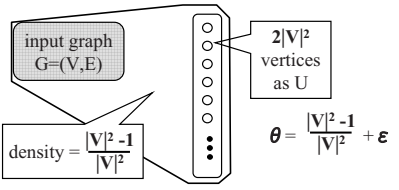


Fig. 2. Construction of the graph for reduction

Problem: Pseudo Clique Existence
For given a graph G, a vertex subset U, and a threshold θ, check whether a pseudo clique K including U exists.

However, as we prove below, this problem is NP-complete. Thus, a straightforward binary partition algorithm will possibly take exponential time in an iteration. Note that this result does not assure that binary partition algorithms will never be output polynomial time even if $P \neq NP$. For example, a good ordering of vertices in V can exist so that the pseudo clique existence in any iteration can be solved in polynomial time. Moreover, even if one iteration takes exponential time in the input size, the total computation time can be polynomial in the output size since the output size can be also exponential in the input size. In general, it is not easy to prove such statements, but also not impossible.

Theorem 1. *The problem pseudo clique existence is NP-complete.*

Proof. The problem obviously belongs to NP, thus we prove NP-hardness by reducing the k clique problem which is to answer whether given graph $G' = (V', E')$ includes a clique of given size k. This problem is NP-complete[7].

We will construct a graph $G = (V, E)$ from G' as follows. Fig. 2 shows an example. Without loss of generality we assume that G' has at least a certain number of vertices, say 10 vertices. Let $V = V' \cup U$ where U is a vertex set of size $2|V'|^2$. We choose two arbitrary vertices z_1 and z_2 from U. The edge set E is defined by

$$E = E' \cup \{(u, v) \mid v \in V', u \in U, u \neq z_1, z_2\} \cup E_U$$

where E_U is an arbitrary edge subset $E_U \subseteq U \times U$ such that $|E_U| = (2|V'|^2 - 1) \times (|V'|^2 - 1)$. Then, the density of $G[U]$ is equal to $(|V'|^2 - 1)/|V'|^2$.

Let K be a subset of V'. If and only if $G[K]$ is a clique, the density of $G[K \cup U]$ is greater than $(|V'|^2 - 1)/|V'|^2$, since the density of $G[\{v\} \cup U]$ is equal to $(|V'|^2 - 1)/|V'|^2$ for any $v \in V'$. For any $K \subset V$ such that $G[K]$ is a clique, the density of $G[K \cup U]$ is determined by the size of K. When K is the empty set or composed of one vertex, its density is $(|V'|^2 - 1)/|V'|^2$, and strictly increases as the increase of the size of K. If the size of K is k, the density is

$$\frac{clq(k) + (2|V'|^2 - 1) \times (|V'|^2 - 1) + k(2|V'|^2 - 2)}{clq(k + |V'|^2)}.$$

Thus, by setting θ to this value, which is the density of $G[U \cup K], K \subseteq V'$ is no less than θ if and only if K is a clique of at least size k in G'. Thus, the problem pseudo clique existence is NP-complete. □

4 Polynomial Delay Enumeration of Pseudo Cliques

In the previous section we showed that a straightforward approach to the pseudo clique enumeration may fail in the sense of the polynomiality. However, the problem can actually be solved in polynomial delay. In this section we describe a polynomial delay algorithm for the non-weighted version of our problem. Necessary modifications for adopting the weighted version is obvious, thus we omitted them. The key to an efficient enumeration is that we construct a tree shaped traversal route on the set of pseudo cliques, and perform a depth first search on the tree, without having the traversal route explicitly on the memory. Such a transversal route can be obtained by looking at an adjacency relation on pseudo cliques spanning on all the pseudo cliques. This technique is called reverse search, which is originally developed by Avis and Fukuda[3].

The idea of reverse search is as follows. We first define a parent for each element to be enumerated, except for specified element called the *root*. The definition of the parent has to be acyclic, that is, each element is not a proper ancestor of itself. Then, the parent-child relation induces a spanning tree rooted at the root on the set of elements to be enumerated. We call the tree the *enumeration tree*. The algorithm traverses the tree in a depth first manner. Reverse search does not need to memorize the previously visit elements in memory space to avoid duplications. It uses an algorithm for listing the children of an element. By finding a child of an element, we can go deeper on the enumeration tree by a recursive call. After we come back from the recursive call, we find the next child and make a recursive call for it. In this way, we can perform the depth first search with memory space linear in the height of the enumeration tree[1].

First we observe the following property to obtain an adjacency relation on pseudo cliques.

Lemma 1. *Let v be a vertex in $G[K]$ with the degree no greater than the average of the degrees of vertices in $G[K]$. The density of $K \setminus \{v\}$ is no less than the density of K.*

Proof. If $|K| = 1$, then $K \setminus \{v\} = \emptyset$, thus the statement holds. Hence, we assume that $|K| \geq 2$. Let F_1 and F_2 be a partition of the set of pairs of vertices in U such that F_1 is the set of pairs of v and another vertex, and F_2 is the set of the remaining pairs. Let $E_1 = E[K] \cap F_1$, and $E_2 = E[K] \cap F_2$. Then, we can observe that the density of U is between $|E_1|/|F_1|$ and $|E_2|/|F_2|$. The density multiplied by $|U| - 1$ is equal to the average degree of vertices in $G[K]$, and thus it is no less than $|E_1|/|F_1|$. Since $E[K \setminus \{v\}] = E[K] \cap F_2$, the density of $K \setminus \{v\}$ It concludes the proof. □

[1] The space complexity of the original reverse search does not depend on the height of the enumeration tree.

From the lemma, we can see that for any pseudo clique K, $K \setminus \{v\}$ is also a pseudo clique for any vertex whose degree is no greater than the average. This introduces an adjacency relation on pseudo cliques. Since any K has such a vertex v, we can remove the vertices of K iteratively until K will be the emptyset, passing through only pseudo cliques. Thus, the adjacency spans all the pseudo cliques. The graph induced by the adjacency is not a tree, thus we introduce parent-child relation to obtain a tree-shaped traverse route.

For a vertex set $K \neq \emptyset$, we define $v^*(K)$ by the minimum degree vertex in $G[K]$. If there are more than one minimum degree vertices, we choose the minimum index one. We define the parent $Prt(K)$ of $K \neq \emptyset$ by $K \setminus \{v^*(K)\}$. From Lemma 1, $Prt(K)$ is a pseudo clique if K is a pseudo clique. Since any K has a unique parent, the graph induced by the parent-child relation forms a tree. In Fig 1, the degrees of vertices in $G[K]$ are $deg_K(3) = deg_K(6) = 2, deg_K(5) = deg_K(9) = 3, deg_K(11) = 4$, thus $v^*(K)$ is vertex 3. The parent of $K = \{3, 5, 6, 9, 11\}$ is $\{5, 6, 9, 11\}$. We will describe an algorithm for traversing the tree.

Now the remaining task is how to find the children of a pseudo clique. For this task, we first observe the following property, immediately obtained from the definition of the parent.

Property 1. For a pseudo clique $K \subseteq V$, K' is a child of K if and only if $K' \setminus K = \{v\}$ and $v = v^*(K')$.

From the property, we can see that K has at most $|V| - |K|$ children, each of which is obtained by adding a vertex v not in K to K. Thus, we can list the children of K by computing the density of $K \cup \{v\}$ and $v^*(K \cup \{v\})$ for each vertex $v \notin K$. In this way, we can list the children thus can enumerate all pseudo cliques. We describe the algorithm as follows, which enumerates all pseudo cliques by calling with G and $K = \emptyset$.

Algorithm. EnumPseudoClique ($G = (V, E)$, K)
1: **output** K
2: **for each** $v \notin K$ **do**
3: **if** $K \cup \{v\}$ is a pseudo clique **then**
4: **if** $K \cup \{v\}$ is a child of K **then** EnumPseudoClique ($G = (V, E)$, $K \cup \{v\}$)

In a straightforward implementation, an iteration of the algorithm takes $O(|V|^2)$ time, thus pseudo cliques can be enumerated $O(|V|^2)$ time for each. The computation time can be reduced by a sophisticated process. We characterize vertices generating children in terms of deg_K, and describe a more efficient algorithm. The density of $K \cup \{v\}$ is $(|E[K]| + deg_K(v))/clq(|K| + 1)$, thus there is a threshold value $\theta(K) = \theta clq(|K| + 1) - |E[K]|$ such that for any vertex v with $deg_K(v) \geq \theta(K)$, $K \cup \{v\}$ is a pseudo clique.

Lemma 2. *Let $K \subseteq V$ be a pseudo clique and v be a vertex not in K. Then, $K \cup \{v\}$ is a pseudo clique if and only if $deg_K(v) \geq \theta(K)$*

We use buckets for $0, ..., |V|-1$ so that the bucket for i stores vertices v satisfying $deg_K(v) = i$. Then, we can take the vertices generating pseudo cliques in constant

time for each. We update the buckets in each iteration and it takes $O(deg(v))$ time for the last added vertex v.

To check whether $v^*(K \cup \{v\}) = v$ or not, we introduce a total order $\prec_U$ for any vertex subset $U \neq \emptyset$, defined by

$$u \prec_U v \quad \Leftrightarrow \quad deg_U(u) < deg_U(v) \quad \text{or} \quad (deg_U(u) = deg_U(v) \quad \text{and} \quad u \leq v),$$

here $u \leq v$ means that the index of u is less than that of v. Then, $v^*(K)$ is the vertex of U satisfying $v^*(K) \prec_U u$ for any other vertex $u \in U$. Then, we have the following lemma.

Lemma 3. *For any pseudo clique K and a vertex v not in K, $K \cup \{v\}$ is a child of K if and only if (1) $K \cup \{v\}$ is a pseudo clique, (2) the tuple $(deg_K(v^*(K)), v^*(K))$ is lexicographically larger than the tuple $(deg_K(v) - 1, v)$, and (3) v is adjacent to any vertex $u \in K, u \prec_K v$.*

Proof. We first observe that for any vertex u, $deg_{K\cup\{v\}}(u) = deg_K(u) + 1$ if v is adjacent to u, and $deg_{K\cup\{v\}}(u) = deg_K(u)$, otherwise. Then, the only if part of the statement is obvious; if (2) is violated, then $v^*(K) \prec_{K\cup\{v\}} v$. Hence we prove the if part. It is sufficient to prove that conditions (2) and (3) lead that $v^*(K \cup \{i\})$ is v. $v^*(K \cup \{i\})$ is v when any vertex $u \in K$ satisfies $u \prec_{K\cup\{i\}} v$.

Suppose that conditions (2) and (3) hold for v. Then, from condition (3), any vertex $u \in K$ satisfying $u \prec_K v$ is adjacent to v. Thus, the vertex u satisfies $deg_{K\cup\{v\}}(u) = deg_K(u) + 1$. From condition (2), the tuple $(deg_{K\cup\{v\}}(u), u)$ is lexicographically larger than $(deg_{K\cup\{v\}}(v), v) = (deg_{K\cup\{v\}}(v), v)$. Thus, the statement holds. □

We show an example in Fig. 1. $K \cup \{v\}$ is a child of K for $v = 1, 2, 4$. Vertex 7 does not satisfy (3), and vertex 10 does not satisfy (2).

If a vertex v satisfies that $K \cup \{v\}$ is a pseudo clique and $v \prec_K v^*(K)$, then $K \cup \{v\}$ is always a child of K. By keeping the above bucket sorted in the order of indices, we can find all such vertices in $O(\log |V|)$ time for each. We note that using a sophisticated heap algorithm which realizes constant time insertion, we can still bound the computation time for update the buckets by $O(deg(v))$. We then explain the way to efficiently find vertices v satisfying that $v^*(K) \prec_K v$ but $K \cup \{v\}$ is a child of K.

Let $L(K)$ be the sequence of first Δ vertices of K sorted by the order $\prec_K$. If K has less than Δ vertices then $L(K)$ includes all vertices in K. For each vertex $v \notin K$, we define $l(v, K)$ by the first vertex in $L(K)$ which is not adjacent to v. We define $l(v, K)$ by $+\infty$ if v is adjacent to all vertices in $L(K)$. In Fig. 1, $L(K) = (3, 6, 5, 9, 11)$, $l(2, K) = 10$, and $l(7, K) = 3$. The following is obvious.

Lemma 4. *Let v be a vertex satisfying that $v^*(K) \prec_K v$ and $K \cup \{v\}$ is a pseudo clique. Then, $K \cup \{v\}$ is a child of K if and only if $v \prec_K l(v, K)$. In particular, v is adjacent to $v^*(K)$.*

We choose the vertices u in $L(K)$ in the order of $\prec_K$ and look at the vertices adjacent to u, then $l(v, k)$ can be computed for all vertices v adjacent to at least

one vertex of $L(K)$. This is done in $O(\min\{\Delta^2, |V| + |E|\})$ time, hence all the vertices v generating children and $v^*(K) \prec_K v$ can be found in $O(\min\{\Delta^2, |V| + |E|\})$ time.

To perform our reverse search, we have to keep $L(K)$ and the buckets in the memory. This requires $O(|V| + |E|)$ space. We need no extra memory for other operations, thus we have the following theorem.

Theorem 2. *For a given graph G and threshold θ, all pseudo cliques can be enumerated in $O(\min\{\Delta^2, |V|+|E|\})$ time for each within $O(|V|+|E|)$ memory. In particular, the delay is $O(\min\{\Delta^2, |V| + |E|\})$.*

We note that the delay can be bounded by the computation time for one iteration by using so called "odd-even output method", described in [13,18]. We modify the algorithm so that in each iteration, the algorithm outputs the solution before generating recursive calls if the depth of the recursion is odd, and after the recursive calls otherwise. By this, during the execution, any three consecutive iterations output at least one solution, thus the delay is reduced to be equal to the computation time for one iteration.

Considering the practice, the estimation of the computation time is too large, since we are usually given a possibly large but sparse graphs thereby the pseudo cliques are small in comparison to the graph sizes. Otherwise the number of pseudo cliques explodes so that we can not handle the output. Thus, there will be few candidates for children, and few vertices adjacent to $v^*(K)$. Thus the complexity we state here is possibly far from the practical computation time.

5 Computational Experiments

We here present the results of computational experiments to show the practical efficiency of our algorithm. The implementation is coded by C, compiled by gcc, and executed in a notebook PC with a Pentium M 1.1GHz processor with 256MB memory with cygwin which is an emulator of Linux environments on Windows. The implementation is a simpler version of our algorithm, which maintains only $deg_K(v)$ for each vertex v, thus the worst computation time for an iteration is $O(|E| + |V|)$. The reason that we did not use the technique to find the children in $O(\log |V|)$ time for each is that in practice we expected that only few vertices satisfy $deg_K(v) = deg_K(v^*(K))$ on average. This was observed in the computational experiments.

We examined several types of graphs as inputs of the implementation, randomly generated graphs and graph taken from real world data. We had three groups of random graphs which are generated in the following three different ways. The first group consists of ordinary random graphs. For each pair of vertices, we connected them by an edge with the same probability 0.1. The second group is of so called locally dense graphs. Consider a necklace sequence obtained by connecting the head and the tail of the vertex sequence $(1, 2, \ldots, |V|)$. For each vertex v, we connected it to each its neighbor with probability 1/2. Here a

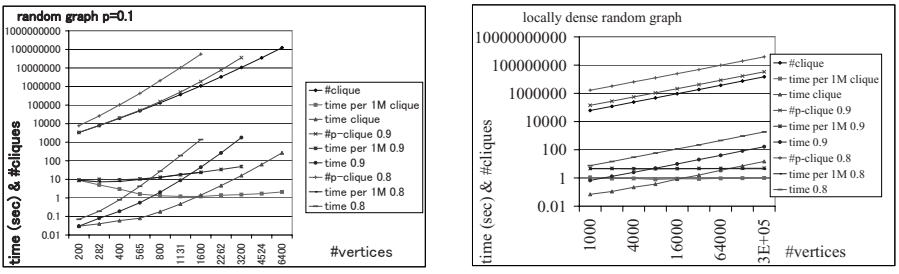


Fig. 3. Results for random graphs (left) and locally dense random graphs (right)

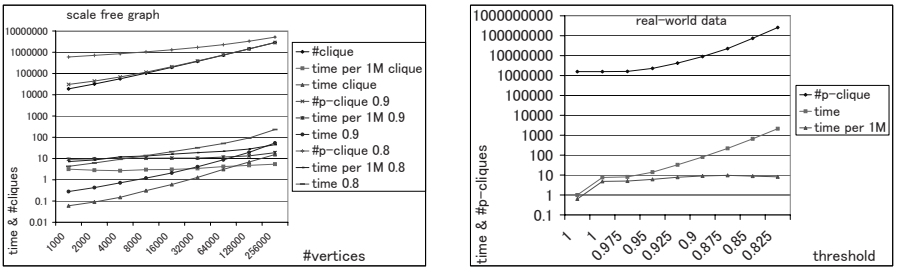


Fig. 4. Results for scale free graphs (left) and co-author graph (right)

neighbor of v is a vertex u with $|u - v| \leq r$, or $|u - v| \geq |V| - r$, for given r. In the experiments we set $r = 20$.

The third was randomly generated scale free graphs. In a scale free graph, the probability that the degree of a vertex is λ is $1/\lambda^{\Gamma}$. Such a distribution is called zip distribution, and such data is said to satisfy power law. The graphs appearing in real world problems are often scale free graphs. Our scale free graphs are generated by starting from a clique of k vertices and adding vertices one-by-one to it, and then connect it to k vertices. The vertices to be connected are chosen randomly such that a vertex is chosen with a probability proportional to its degree. A graph generated in this way is known to be scale free. The graph tends to have few locally dense structures which we can see many in real world data, thus the average size of cliques in this graph is often small.

We run the implementations for these graphs with the thresholds $\theta = 0.8$ and $\theta = 0.9$. Since we could not find any implementations for the pseudo clique enumeration problem, we compare the performance to that of an ordinary backtracking clique enumeration algorithm, which maintains $deg_K(v)$ to find candidates for the addition. Since the clique enumeration is a special case of our problem, it can be considered as a kind of upper bound of the performance of the pseudo clique enumeration.

The results of the experiments are shown in figures. The left side of Fig. 3 shows the results for ordinary random graphs, and the right side of Fig. 3 is for locally

dense random graphs The left side of Fig. 4 is for scale free graphs. The horizontal axis is the number of vertices in the input graphs, and the vertical axis is for the computation time, computation time for 1 million (pseudo) cliques, and the number of output (pseudo) cliques. All these are shown in log scales. The line almost horizontal in the figures display the computation time per one million pseudo cliques. The computation time for each pseudo clique does not change with the change in the threshold value θ, and does not differ very much from that of the clique enumeration. This means that the performance of the pseudo clique enumeration is close to optimal, thus in the practice a high performance is expected.

The computation time is increasing with the increase of the graph sizes in the scale free graphs and slightly increasing in random graphs. This is possibly because of the increase in the average degree, and the decrease in the ratio of children in the candidates of children. When we set threshold value θ to a small value, the number of vertices v satisfying $deg_K(v^*(K)) \leq deg_K(v) \leq deg_K(v^*(K)) + 1$ increases. Since scale free graphs has a kind of locality, few of them will be the children of K, on average.

For all the results, the diagonal lines represents the numbers of pseudo cliques and cliques. For the random graphs, the ratio of these three values increases with the increase in graph size. This could be because of the increase in the average degree. It is interesting to note that the ratio does not change much in locally dense random graphs, and is reduced for the scale graphs.

The right side of Fig. 4 is the result for the graph taken from the real world data. The graph is a co-author graph[19] such that each vertex is a researcher and two researchers are connected if they have a joint paper. The number of vertices is about 30,000 and the number of edges is about 125,000. It is known that the graph is a scale free graph. For this graph we observe the results by changing the threshold θ. The leftmost point indicates the computation time of the clique enumeration, thus it is faster than the others, but not different much from that of the pseudo clique enumeration. The computation time does not seem to depend on the threshold value.

6 Conclusion

In this paper we addressed the problem of finding dense structures from a graph. The density is given by the ratio of the existing edges compared to a complete graph, and we define a pseudo clique as a dense structure by a subgraph with density no less than the given threshold value. In this term we define our problem of enumerating all pseudo cliques of given a graph and a threshold.

We first showed that it is not easy to get polynomial time algorithm by straightforward approach since in this way we encounter an NP-complete problem. On the other hand, we show that any pseudo clique has a proper subset being a pseudo clique with one fewer vertices. This induces a relation spanning all pseudo cliques. Using the relation we developed a reverse search algorithm whose delay is $O(\min\{\Delta^2, |E| + |V|\})$, thus computation time for each pseudo clique is $O(\min\{\Delta^2, |E| + |V|\})$.

Recently, it has become popular to use dense structures to represent related objects. One of the problems on practice is that the number of output solutions is often larger than that of the cliques. The "maximal" pseudo clique enumeration may help, but it is not straightforward to introduce maximality because the family of pseudo cliques does not satisfy the monotone property. Detailed characterizations of the dense structures which would allow us to develop efficient algorithms are important for applications, and an interesting open problem.

Acknowledgment

This research was partly supported by the Ministry of Education, Science, Sports and Culture, Grant-in-Aid for Specially Promoted Research, 18017015 for developing high-speed high-quality algorithms for analyzing huge genome database.

References

1. Aslam, J., Pelekhov, K., Rus, D.: A Practical Clustering Algorithms for Static and Dynamic Information Organization. In: SODA 1999, ACM Press, New York (1999)
2. Arora, S., Karger, D., Karpinski, M.: Polynomial time approximation schemes for dense instances of NP-hard problems. In: Proc. ACM Symp. on Theory of Comp., pp. 284–293 (1995)
3. Avis, D., Fukuda, K.: Reverse Search for Enumeration. Discrete Applied Mathematics 65, 21–46 (1996)
4. Feige, U., Peleg, D., Kortsarz, G.: The dense k-subgraph problem. Algorithmica 29, 410–421 (2001)
5. Fukuda, K., Matsui, T.: Finding All Minimum-Cost Perfect Matchings in Bipartite Graphs. Networks 22, 461–468 (1992)
6. Fujisawa, K., Hamuro, Y., Katoh, N., Tokuyama, T., Yada, K.: Approximation of Optimal Two-Dimensional Association Rules for Categorical Attributes Using Semidefinite Programming. In: Arikawa, S., Furukawa, K. (eds.) DS 1999. LNCS (LNAI), vol. 1721, pp. 148–159. Springer, Heidelberg (1999)
7. Garey, M.R., Johnson, D.S., Stockmeyer, L.: Some Simplified NP-complete Problems. In: Proc. ACM Symp. on Theory of Comp., pp. 47–63 (1974)
8. Gibson, D., Kumar, R., Tomkins, A.: Discovering Large Dense Subgraphs in Massive Graphs. In: Proc. VLDB, pp. 721–732 (2005)
9. Haraguchi, M., Okubo, Y.: A method for clustering of web pages with pseudo-clique search. In: Jantke, K.P., Lunzer, A., Spyratos, N., Tanaka, Y. (eds.) Federation over the Web. LNCS (LNAI), vol. 3847, pp. 59–78. Springer, Heidelberg (2006)
10. Hu, H., Yan, X., Huang, Y., Han, J., Zhou, X.J.: Mining Coherent Dense Subgraphs Across Massive Biological Networks for Functional Discovery. Bioinformatics 21, 213–221 (2005)
11. Kumar, R., Raghavan, P., Rajagopalan, S., Tomkins, A.: Extracting Large-Scale Knowledge Bases from the Web. In: Proc. VLDB, pp. 639–650 (1999)
12. Makino, K., Uno, T.: New Algorithms for Enumerating All Maximal Cliques. In: Hagerup, T., Katajainen, J. (eds.) SWAT 2004. LNCS, vol. 3111, pp. 260–272. Springer, Heidelberg (2004)

13. Nakano, S., Uno, T.: Constant Time Generation of Trees with Specified Diameter. In: Hromkovič, J., Nagl, M., Westfechtel, B. (eds.) WG 2004. LNCS, vol. 3353, pp. 33–45. Springer, Heidelberg (2004)
14. Palla, G., Derenyi, I., Farkas, I., Vicsek, T.: Uncovering the Overlapping Community Structure of Complex Networks in Nature and Society. Nature 435, 814–818 (2005)
15. Read, R.C., Tarjan, R.E.: Bounds on Backtrack Algorithms for Listing Cycles, Paths, and Spanning Trees. Networks 5, 237–252 (1975)
16. Tomita, E., Tanaka, A., Takahashi, H.: The worst-case time complexity for generating all maximal cliques and computational experiments. Theoretical Computer Science 363, 28–42 (2006)
17. Uno, T.: Algorithms for Enumerating All Perfect, Maximum and Maximal Matchings in Bipartite Graphs. In: Leong, H.-V., Jain, S., Imai, H. (eds.) ISAAC 1997. LNCS, vol. 1350, pp. 92–101. Springer, Heidelberg (1997)
18. Uno, T.: Two General Methods to Reduce Delay and Change of Enumeration Algorithms, National Institute of Informatics (in Japan) Technical Report, 004E (2003)
19. Warner, S.: E-prints and the Open Archives Initiative. Library Hi Tech 21, 151–158 (2003)
20. Zhang, Y., Chu, C.H., Ji, X., Zha, H.: Correlating Summarization of Multisource News with k Way Graph Biclustering. ACM SIGKDD Explo. Newslett. arch. 6, 34–42 (2004)

Fast Adaptive Diagnosis with a Minimum Number of Tests

Samuel Guilbault and Andrzej Pelc*

Département d'informatique, Université du Québec en Outaouais, Gatineau, Québec J8X 3X7, Canada
samuel.guilbault@hotmail.com, pelc@uqo.ca

Abstract. We study adaptive system-level fault diagnosis for multiprocessor systems. Processors can test each other and later tests can be scheduled on the basis of previous test results. Fault-free testers correctly identify the fault status of tested processors, while faulty testers can give arbitrary test results. The goal is to identify correctly the status of all processors, assuming that the number of faults does not exceed a given upper bound t, where n is the number of processors. Tests involving disjoint pairs of processors can be performed simultaneously in one round.

Two most important measures of quality of a diagnosis algorithm are its worst-case cost (the number of tests used) and time (the number of rounds used). It is known that the optimal worst-case cost of a diagnosis algorithm is $n + t - 1$. However, the known algorithms of this cost use time $\Theta(n)$. We present an algorithm with optimal cost $n + t - 1$ using time $O(\log t)$, provided that the upper bound t on the number of faults satisfies $t(t + 1) \leq n$. Hence, for moderate numbers of faults which we assume, our algorithm achieves exponential speed-up, compared to the previously known diagnosis algorithms of optimal cost.

1 Introduction

As the size of multiprocessor systems grows they become increasingly vulnerable to component failures. This in turn enhances interest in the issue of reliability of such systems. One of the major problems in this field, known as the fault diagnosis problem, is to precisely locate all faulty processors in the system, i.e., to answer the question which processors are faulty and which are fault free. The classic approach to fault diagnosis was originated by Preparata, Metze and Chien [13]. Processors perform tests on each other, and diagnosis is based on the collection of test results. It is assumed that fault-free processors always give correct test results, while tests conducted by faulty processors are totally unpredictable: A faulty tester can output any test result, regardless of the status of the tested processor. The fault-status of a processor does not change during testing and diagnosis. In [13] a worst-case scenario is adopted: it is assumed

* Research partly supported by NSERC discovery grant and by the Research Chair in Distributed Computing at the Université du Québec en Outaouais.

T. Tokuyama (Ed.): ISAAC 2007, LNCS 4835, pp. 415–426, 2007.

that at most t processors are faulty and that they are placed in locations most disadvantageous for the diagnosis process.

In early stages, research on fault diagnosis mostly focused on *one-step*, or *nonadaptive* diagnosis, already defined in [13]. In this type of diagnosis it is assumed that all tests are determined in advance and they cannot be rescheduled during the diagnosis process. Nakajima [10] was the first to modify this assumption. He proposed a new approach called *adaptive* diagnosis in which a test is determined only after seeing the results of previous ones. The flexibility of this approach increases the efficiency of diagnosis. In [8,14,2,3,1] the parallel time (number of rounds) of adaptive diagnosis was investigated, assuming that only tests involving disjoint pairs of processors can be conducted in the same round. It was shown that while locating $t < n/2$ faults requires worst-case time at least t in the nonadaptive setting, adaptive diagnosis can locate less than $n/2$ faults among n processors in constant time. On the other hand, Blecher [5] showed that the *cost*, i.e., the number of tests required in the worst case to identify $t < n/2$ faults, decreases from tn for nonadaptive diagnosis to $n + t - 1$ in the adaptive setting. He also showed that the latter is a lower bound for the worst-case cost of adaptive diagnosis. However, the algorithm of Blecher, while optimal from the point of view of the number of tests, is very slow: many tests are scheduled sequentially and the time of the algorithm can be as large as $\Theta(n)$. The same is true for the cost-optimal diagnosis algorithm for $(2t - 1)$-connected graphs from [11]. On the other hand, constant time algorithms from [2,3,1] are not cost-optimal. This raises the question of whether it is possible to maintain the cost-optimality of adaptive fault diagnosis, i.e., use only $n + t - 1$ tests, at the same time speeding up significantly the diagnosis process. Doing this is the goal of the present paper.

Our results. We present an adaptive diagnosis algorithm for n-processor systems, with optimal cost $n + t - 1$ and using time $O(\log t)$, provided that the upper bound t on the number of faults satisfies the condition $t(t + 1) \leq n$. Hence, for moderate numbers of faults which we assume, our algorithm achieves exponential speed-up, compared to the previously known diagnosis algorithms of optimal cost. For small values of t the speed-up is even more significant: indeed, if t is logarithmic in n we get double-exponential speed-up, and for constant t our algorithm uses constant time, while the previous optimal-cost algorithms still use time $\Theta(n)$.

It remains open whether it is possible to perform logarithmic-time optimal-cost t-diagnosis for any upper bound $t < n/2$ on the number of faults. Another interesting question is whether $\Theta(\log t)$ is a lower bound on the time of optimal-cost diagnosis.

Related work. The PMC model described above and introduced in [13] was the first analytic model for fault diagnosis. Many of its variations have been subsequently studied in the literature (see the survey [6], where further bibliography can be found). One of the important modifications of this model was replacing the assumption of the fixed upper bound on the number of faults and

their worst case location by a probabilistic approach where faults are assumed to occur randomly (see the survey [9] and the bibliography therein). Another modification, introduced in [10], was replacing one-step (nonadaptive) diagnosis by the adaptive approach. This more flexible testing method enabled to drastically cut diagnosis time. The first result in this direction was obtained in [8], followed by [14], and by [2] where constant time was achieved for any number of faults smaller than $n/2$. Then [3] and [1] further decreased the number of diagnosis rounds. However the above fast diagnosis algorithms were not cost-optimal. The optimal cost of adaptive diagnosis was achieved in [5], under the assumption that all processors can test each other. However this algorithm used $\Theta(n)$ rounds of testing. Adaptive diagnosis assuming that processors fail randomly has been studied in [12]. Another line of research concerned the case when tests that may be conducted are restricted to some graph. For the k-hypercube, optimal-cost diagnosis was achieved in [4] assuming an upper bound k on the number of faults. On the other hand, fastest possible k-diagnosis of the k-hypercube, lasting only 3 rounds, was achieved in [7]. In [11] optimal-cost t-diagnosis was presented for arbitrary $(2t-1)$-connected graphs. The authors provided also optimal-cost diagnosis for cube-connected cycles.

2 Terminology and Preliminaries

Throughout the paper n denotes the number of processors in the system, and t denotes the upper bound on the number of faults. We assume that $t(t+1) \leq n$. Unless explicitly stated otherwise, logarithms are with base 2.

Consider a system U of n processors that can test each other. For processors $u, v \in U$, the ordered pair (u, v) represents the test performed by u on v. In this situation u is called the *tester* and v is called the *tested processor*. The outcome of a test (u, v) is 1 (0), if u evaluates v as faulty (fault-free). According to the PMC model from [13], the outcome of (u, v) is 0 if both u and v are fault free, and it is 1 if u is fault free and v is faulty. Thus fault-free testers always give correct test results. No assumptions are made about the outcome of tests involving a faulty tester. A test with outcome 0 (1) will be called a *0-arrow* (*1-arrow*). In this case the tester is called the beginning and the tested processor is called the end of this 0-arrow (1-arrow). A *0-line* is a sequence $(u_0, \ldots, u_k)$ of processors, such that all tests (u_i, u_{i+1}), for $0 \leq i < k$, were performed and are 0-arrows. A *spike* is a (possibly empty) 0-line $(u_0, \ldots, u_k)$ followed by a single 1-arrow beginning at u_k. (Thus a single 1-arrow is also a spike.)

A *test assignment* is a collection of tests (u, v) for some pairs of processors. It can be modeled as a directed graph $T = (U, A)$, where $(u, v) \in A$. In the case of adaptive diagnosis, a test assignment is constructed dynamically, the next test being a function of previous test results. The collection of all test results for a test assignment T is called a *syndrome*. Formally, a syndrome is any function $S : A \longrightarrow \{0, 1\}$. The set of all faulty processors in the system is called a *fault set*. This can be any subset of U. A syndrome S is said to be *compatible* with a fault set $F \subseteq U$ if, for any $(u, v) \in A$, such that $u \in U \setminus F$, $S(u, v) = 1$ iff $v \in F$.

This corresponds to the assumption that fault-free testers always give correct test results. Since faulty testers can give arbitrary test results, any syndrome compatible with a fault set F can occur when faulty processors in the system are exactly those in F.

Consider an adaptive diagnosis algorithm. Since the test assignment $T = (U, A)$ is constructed adaptively, on the basis of previous test results, at the end of the algorithm a syndrom $S : A \longrightarrow \{0, 1\}$ is obtained. The algorithm is a correct t-diagnosis if there is at most one fault set F of size at most t, compatible with this syndrome. This means that the algorithm designs a test assignment in such a way that any configuration of at most t faults can be correctly diagnosed on the basis of the obtained syndrome, regardless of the behavior of faulty testers. It follows from [13] that if a correct t-diagnosis exists then the number n of nodes must exceed $2t$.

An important measure of efficiency of an adaptive diagnosis algorithm is its worst-case *cost*, i.e., the number of tests that it uses in the worst case. Blecher [5] proved that the minimum cost of a t-diagnosis for n processors is $n + t - 1$, provided that the necessary condition $t < n/2$ is satisfied.

Another measure of performance of adaptive fault diagnosis is the *time*, i.e., the number of *rounds* used for testing. It is assumed that tests involving disjoint pairs of processors can be carried out in parallel, in the same round, while tests in which at least one processor is common must be scheduled in different rounds. In [1,2,3] it was shown how to diagnose a n-processor system with at most $t < n/2$ faults, in constant time, independent of n and t.

We will use the following propositions. The first is an immediate consequence of the assumptions of the model.

Proposition 1. *Let $(u_0, \ldots, u_k)$ be a 0-line. If the processor u_i is fault-free then so are all processors u_j, for $i < j \leq k$. If the processor u_i is faulty then so are all processors u_j, for $0 \leq j < i$.*

The second proposition is a well know algebraic property. It follows from the fact that the geometric mean of positive numbers does not exceed their arithmetic mean.

Proposition 2. *Let $x_1, \ldots, x_k$ be positive reals such that $x_1 + \cdots + x_k = x$. Then $\log x_1 + \cdots + \log x_k \leq k \log(x/k)$.*

3 Algorithm Fast-Minimum-Cost-Diagnosis

3.1 An Auxiliary Procedure

We start with the following procedure that will be used in our algorithm. The input of the procedure is a 0-line $(u_0, \ldots, u_k)$ and a processor v different from all u_i and known to be fault free. The aim of the procedure is to diagnose all processors u_i using $O(\log k)$ tests. The idea of the procedure comes from the algorithm to find an unknown integer using comparison queries: first confine the

Procedure Fast-Line-Scan

Phase 1: jumping

perform tests (v, u_{2^j-1}), for $j = 0, 1, 2, \ldots, \lfloor \log k \rfloor$, until result 0 is obtained
if all results are 1 **then** perform test (v, u_k)
    if the last result is 1 **then** diagnose all processors as faulty
    else $a := 2^{\lfloor \log k \rfloor}$; $b := k - 1$
else
    $r :=$ the index for which (v, u_{2^r-1}) gives result 0
    diagnose all processors u_i, for $2^r - 1 \leq i \leq k$, as fault free
    diagnose all processors u_i, for $0 \leq i \leq 2^{r-1} - 1$, as faulty
    $a := 2^{r-1}$; $b := 2^r - 2$

Phase 2: binary search in $[u_a, u_b]$

$m := \lfloor (a + b)/2 \rfloor$
while $a < b$ **do**
    perform test (v, u_m)
    if the result is 0 **then**
        diagnose all processors $u_m, \ldots, u_b$ as fault free
        $b := m - 1$
    if the result is 1 **then**
        diagnose all processors $u_a, \ldots, u_m$ as faulty
        $a := m + 1$
    $m := \lfloor (a + b)/2 \rfloor$

unknown integer within an interval $[2^i, 2^{i+1}]$ by performing jumps 2^j and then locate it within this interval by binary search.

The following lemma is a consequence of Proposition 1.

Lemma 1. *Procedure Fast-Line-Scan correctly diagnoses the 0-line* $(u_0, \ldots, u_k)$.

The next lemma establishes the complexity of the procedure.

Lemma 2. *Procedure Fast-Line-Scan uses at most* $2 \log t + 2$ *tests if* t *is an upper bound on the number of faults in the 0-line* $(u_0, \ldots, u_k)$.

Proof. Phase 1 uses at most $\log t + 2$ tests and Phase 2 uses at most $\log t$ tests.

3.2 Overview of the Algorithm

Note that it is enough to describe and analyze the algorithm for t sufficiently large. Below some threshold the algorithm of Blecher can be used. It uses $n+t-1$ tests and its time will not affect the asymptotic estimate $O(\log t)$.

Algorithm Fast-Minimum-Cost-Diagnosis is divided into 4 phases. The aim of Phase 1 is to partition all processors into 0-lines and spikes. The aim of Phase 2 is to identify at least $t + 1$ fault-free processors. This is done by constructing

at least one cycle of length at least $t+1$ which is a 0-line and whose closing test is also a 0-arrow (i.e., all clockwise tests in this cycle are 0-arrows). No such cycle can contain faulty processors. (In one special case a cycle with the above properties cannot be identified but then the diagnosis can be easily finished.) In Phases 3 and 4 the processors diagnosed as fault free in Phase 2 are used to test other processors. In Phase 3 short 0-lines and spikes are diagnosed in parallel, in each of them proceeding from the beginning to end. Finally, in Phase 4 long 0-lines and spikes are diagnosed in parallel, in each of them proceeding from the beginning to end, at some point possibly calling Procedure Fast-Line-Scan for some of them in parallel, in order to maintain logarithmic diagnosis time.

3.3 Description of the Algorithm

Phase 1: Forming segments
Partition the set U of processors into pairwise disjoint sets $A_1, A_2, \ldots, A_t$, of sizes differing by at most 1. Let $x = \lceil \log_{32/31} t \rceil$. Partition each set A_i into pairwise disjoint sets of sizes between $2x$ and $3x$. Call these sets *segments*. Let t (and hence also n) be sufficiently large that each set A_i contains at least one segment. Let $S_1, \ldots, S_m$ be an enumeration of all segments. Enumerate the elements of each S_j as follows: $v_1^j, v_2^j, \ldots$.

In at most $3x$ rounds perform the following tests. In round r perform all tests (v_r^j, v_{r+1}^j), for all $j \leq k$ in parallel, skipping those tests for which v_r^j is the end of a 1-arrow obtained in round $r-1$. Call a segment *pure* if it is a 0-line. In any set A_i containing at least two pure segments, order all its pure segments and perform in all these sets in parallel (in one round) all tests (u, v), where u is the last processor of a pure segment and v is the first processor of the next pure segment.

Phase 2: Forming cycles using pure segments
In all sets A_i forming a single 0-line $(u_0, \ldots, u_k)$ at the end of Phase 1, perform in parallel (in one round) all tests (u_k, u_0). For all sets A_i for which this test is a 0-arrow, diagnose all processors in A_i as fault free.

If there is a 1-arrow (u, v) in every set A_i after performing the above tests (call this case Special), then diagnose all processors in A_i other than u and v as fault free. If u was already tested, diagnose it as fault free and diagnose v as faulty. Otherwise, use some processor $w \in A_i$ already diagnosed as fault free to test u. All these tests should be done in all A_i in parallel in 1 round. If the resulting test is a 1-arrow, diagnose u as faulty and v as fault free. If the resulting test is a 0-arrow, diagnose u as fault free and v as faulty.

Lemma 3. *1. If the Special Case occurs in Phase 2 then diagnosis is completed. It uses at most n tests and $O(\log t)$ rounds.*
2. If the Special Case does not occur in Phase 2 then all processors diagnosed as fault free at the end of Phase 2 are indeed fault free. At least $t+1$ processors are diagnosed as fault free.

Proof. 1. If the Special Case occurs in Phase 2, there is exactly one 1-arrow and exactly one fault in each set A_i. Let (u, v) be the unique 1-arrow in A_i.

Since the faulty node must be either u or v, all other nodes are fault free. One of them is used to test u, hence the test result is reliable. Diagnosis of v follows. If u was already tested, the test must have been a 0-arrow, hence u must be fault free (otherwise there would be at least two faulty nodes in this set A_i). If u has not been tested yet, one more test is used in this set A_i. In any case, the total number of tests is at most n. The number of rounds is at most $3x + 2 \in O(\log t)$.

2. All sets A_i have size at least $t+1$. If the Special Case does not occur in Phase 2 then a processor identified as fault free was in a cycle of 0-arrows of size at least $t + 1$. Any such processor must be indeed fault free (a faulty processor in such a cycle would imply that all processors in the cycle are faulty, thus contradicting the upper bound t). Since the Special Case did not occur, at least one set A_i contains only fault-free processors. In all these sets cycles of 0-arrows were obtained and hence all processors in these sets were diagnosed as fault free. This gives at least $t + 1$ fault-free processors.

In view of Lemma 3 we may assume in the sequel that the Special Case does not occur in Phase 2. Let X be the set of processors diagnosed as fault free at the end of Phase 2.

Phase 3: Testing non-pure segments
In Phase 3 processors from X are used as testers to test segments that were not pure in Phase 1. (Since all processors in X are fault free, their test results are reliable.) After Phase 1, any non-pure segment is partitioned into a pairwise disjoint collection of spikes and possibly one 0-line. The first processor of each spike and 0-line has not been tested yet. Some 0-lines and spikes in such a segment may be very short, in fact a spike may be even reduced to a single 1-arrow. All of them have length at most $3x$. Since each spike must contain at least one faulty processor and there is at most one 0-line per segment (its final part), the total number of spikes and 0-lines that come from non-pure segments is at most $2t$ and hence there are enough processors in X to test one processor from each such spike and 0-line in at most two rounds. Let $T_1, \ldots, T_m$ be all 0-lines and spikes resulting from non-pure segments. Consider two cases.

Case 1. There was at least one pure segment in Phase 1 not yet diagnosed.

In this case testing of non-pure segments is done as follows. In the first round of Phase 3 all first processors of $T_1, \ldots T_m$ are tested. If a given test is a 0-arrow, the entire corresponding T_i is diagnosed: the tested processor and all subsequent ends of 0-arrows as fault free and the end of the final 1-arrow (if any) as faulty. If a given test is a 1-arrow, the tested processor is diagnosed as faulty. Lines T_i which are not yet entirely diagnosed, pass to the second round. In the rth round of Phase 3, the rth processors of each line that passed to the rth round are tested and diagnosis is as above, with lines corresponding to 1-arrows passing to round $r + 1$. In the final round all last processors of all surviving lines are tested and diagnosed.

Case 2. There was no pure segment in Phase 1 not yet diagnosed.

In this case testing of non-pure segments is done slightly more carefully because at least one test has to be saved. In all rounds tests are performed as in Case 1, except for testing of the last processor of each line. These tests are delayed to the last round. Let t' be the number of faulty processors diagnosed until this round. If $t' < t$ then all remaining processors are tested and diagnosed in the last round. If $t' = t$ then no further tests are performed.

Phase 4: Testing remaining segments

In Phase 4 processors from X are used to test processors that are not yet diagnosed at the end of Phase 3. Such non diagnosed processors exist only if there was at least one pure segment in Phase 1. Consider a set A_i containing such a segment. The last round of tests in Phase 1 joined all pure segments of A_i in a line of processors where the first processor has not been tested and each processor except the last one has tested the next processor in the line. Moreover, there is at least one 1-arrow, and between any two 1-arrows there are more than x consecutive 0-arrows.

We now partition each such line of processors into pairwise disjoint spikes and a 0-line as follows. The first spike starts at the first processor of the line and ends at the end of the first 1-arrow. The second spike starts at the processor following it and ends at the end of the second 1-arrow. In general, the jth spike starts at the processor following the end of the the $(j-1)$th 1-arrow and ends at the end of the jth 1-arrow. The unique 0-line starts at the processor following the end of the last 1-arrow and ends at the last processor of the line.

Note that the total number of spikes thus obtained is at most t because there must be at least one faulty processor in every spike. Moreover, there is exactly one 0-line per set A_i remaining undiagnosed. In each such set there must have been at least one 1-arrow and hence the number of these 0-lines is also at most t. Any spike or 0-line obtained above will be called a *string*. Hence the total number of strings is at most $2t \leq 2|X|$, and consequently there are enough processors in X to test one processor per string in at most two rounds. This is how testing proceeds in Phase 4. Also note that the length of each string is at least $2x$.

Suppose that the number of processors diagnosed as faulty before Phase 4 is t'. Let $\tau = \tau_1 = t - t'$. In the first round of Phase 4 the first processor in each string is tested. If a given test is a 0-arrow, the entire corresponding string is diagnosed: the tested processor and all subsequent ends of 0-arrows as fault free and the end of the final 1-arrow (if any) as faulty. If a given test is a 1-arrow, the tested processor is diagnosed as faulty. We denote by Z_1 the set of end segments of strings not yet diagnosed after round 1, by b_1 the number of processors diagnosed as faulty in round 1 and we put $\tau_2 = \tau_1 - b_1$. More generally, let Z_r be the set of end segments of strings not yet diagnosed after round r, let b_r be the number of processors diagnosed as faulty in round r and $\tau_r = \tau_{r-1} - b_{r-1}$. As long as $b_r \geq \tau_r/32$ and $\tau_r > 0$, in round $r+1$ the first processor of each end segment from Z_r is tested. Let r^* be the first round for which $b_{r^*} < \tau_{r^*}/32$, if such a round exists. Then, starting in round r^*+1 the Procedure Fast-Line-Scan is called in parallel for all end segments in Z_{r^*}. If the end segment is a spike then

if some processor in the 0-line forming it is diagnosed as fault free then the last processor (the end of the final 1-arrow) is diagnosed as faulty; if all processors in the 0-line are diagnosed as faulty then one additional test is used to diagnose the end of the final 1-arrow.

4 Analysis

In this section we prove that Algorithm Fast-Minimum-Cost-Diagnosis is correct and we analyze its cost and time.

Theorem 1. *Algorithm Fast-Minimum-Cost-Diagnosis performs correct diagnosis of a system of n processors, provided that the upper bound t on the number of faults satisfies $t(t+1) \leq n$.*

Proof. It follows from Lemma 3 that diagnosis is correct until the end of Phase 2 and that $|X| \geq t+1$. Correctness of all diagnostic decisions in Phases 3 and 4, except in the possible calls of Procedure Fast-Line-Scan, follows from the fact that fault-free testers are reliable: all these decisions have the form: a 0-line starting from a fault-free processor is fault free and a 1-arrow with a fault-free beginning must have a faulty end. Correctness of diagnostic decisions in the calls of Procedure Fast-Line-Scan follows from Lemma 1. The fact that all processors are diagnosed follows from the formulation of the algorithm.

The next theorem establishes the cost and the time of the algorithm.

Theorem 2. *For n-processor systems, Algorithm Fast-Minimum-Cost-Diagnosis uses $O(\log t)$ rounds and $n+t-1$ tests in the worst case, provided that the upper bound t on the number of faults satisfies $t(t+1) \leq n$.*

Proof. We first estimate the number of rounds used by the algorithm. Phase 1 lasts at most $3x+1$ rounds. Phase 2 lasts at most 2 rounds. Phase 3 lasts at most $3x+2$ rounds. Consider Phase 4. The condition $b_r \geq \tau_r/32$ can be satisfied at most for $x = \lceil \log_{32/31} t \rceil$ rounds because during each such round at least $1/32$ of the remaining pool of faults is revealed. Hence, if Procedure Fast-Line-Scan is not called, Phase 4 lasts at most x rounds. If Procedure Fast-Line-Scan is called, we have $r^* \leq x$ and the procedure lasts at most $2\log t + 2$ rounds, in view of Lemma 2. Procedure Fast-Line-Scan is applied to end segments from Z_{r^*} which are either 0-lines or spikes. For those end segments that are 0-lines, Procedure Fast-Line-Scan is called without any change, for those that are spikes, it is called for the 0-line part of the spike (all processors of the spike except the last one) and the last processor is tested separately in the final round. In this case, one round has to be added. Hence the total number of rounds is at most $7x + 2\log t + 8$, hence it is $O(\log t)$, in view of the definition of x.

We now proceed with the more difficult estimate of the number of tests performed by the algorithm. Here we will use the method of charging tests either to nodes or to discovered faults. Each test will be charged either to a node that was not charged previously, or to a fault diagnosed as a consequence of this test.

This will guarantee the upper bound $n + t$. In order to show the desired tight bound $n + t - 1$ we will have to show that in every case either a node or a fault remains uncharged.

First consider Phases 1 and 2. In these phases no faults are diagnosed and all tests are charged to nodes. More precisely, each performed test is charged to the node tested during it. As a consequence, in sets A_i where a cycle of 0-arrows was constructed (and hence all processors were diagnosed as fault free), all nodes are charged. In each remaining set A_i, all pure segments were joined in a single line and hence the only uncharged node is the beginning of this line. Segments that were not pure were partitioned into 0-lines and spikes, the beginning of each of them being yet uncharged.

In Phase 3 there are two cases. In Case 1, when there was at least one pure segment in Phase 1 not yet diagnosed by the start of Phase 3, our analysis may be more loose because the required saving of one test will be done in Phase 4. So in this case we charge as follows. Whenever a test is a 1-arrow, we charge this test to the fault at the end of it. Whenever a fault is a 0-arrow and the tested processor is in a spike, but not the end of it, we charge this test to the fault at the end of the spike. Whenever a fault is a 0-arrow and the tested processor is either in a 0-line or it is the last processor of a spike, we charge the test to the node which is the beginning of this 0-line or spike. This covers all possible tests in Phase 3.

In Case 2, when there was no pure segment in Phase 1 not yet diagnosed by the start of Phase 3, the analysis in Phase 3 must be done more precisely, as in this case Phase 3 is the final one and we must guarantee that some node or (potential) fault remains uncharged. We do charging as before, except in the last round when the last processor of each surviving 0-line and spike is tested. Let t' be the number of faulty processors diagnosed until this round. If $t' < t$ then all remaining processors are tested in the last round. We charge each of these tests to the processor beginning the respective 0-line or spike. Thus the total number of charges is at most $n + t'$ in this case. Since $t' < t$, we guarantee the upper bound $n + t - 1$. If $t' = t$ then no further tests are performed. Since at least one 0-line or spike surivived to this final round, the beginning of this 0-line or spike remains uncharged and we have at most $t + (n - 1)$ charges made.

Finally we analyze Phase 4. This phase is performed only if Case 1 in Phase 3 occured. There are two possibilities in Phase 4. First suppose that Procedure Fast-Line-Scan was never called. This means that all faults were diagnosed by round x of Phase 4. Let ρ be the last phase for which $\tau_\rho > 0$. Since strings diagnosed in Phase 4 have length at least $2x$, none of the last processors in these strings is tested in round ρ. Charging is done as in Case 1 of Phase 3. Since $\tau_\rho > 0$ and ρ is the last round of the algorithm in this case, it is impossible to diagnose all processors as fault free in this round. Hence either some test is a 1-arrow, in which case the first processor of the corresponding string remains uncharged, or all tests are 0-arrows but at least one of them corresponds to a spike, in which case this test is charged to the fault in the last processor of this spike, and the first processor of the spike remains uncharged. This implies that

if Procedure Fast-Line-Scan was never called, the number of tests is at most $n + t - 1$.

Next suppose that Procedure Fast-Line-Scan has been called starting in round $r^* + 1$, in parallel for all end segments in Z_{r^*}. Hence we have $b_{r^*} < \tau_{r^*}/32$. Let $F_1, \ldots, F_k$, for $k \leq b_{r^*}$, be the end segments in Z_{r^*}, remaining to be diagnosed. Let λ' be the number of faults diagnosed until the end of round r^* and let $\lambda = t - \lambda'$. Since $\lambda = \tau_{r^*} - b_{r^*} > 31\tau_{r^*}/32$ and $k \leq b_{r^*} < \tau_{r^*}/32$, we have $\lambda = \alpha k$, for some $\alpha > 31$. Each F_i is either a 0-line or a spike. As mentioned above, for those F_i that are 0-lines, Procedure Fast-Line-Scan is called without any change, and for those that are spikes, it is called for the 0-line part of the spike and the last processor is tested separately. Hence if f_i is the number of faults in F_i, the number of tests used to diagnose this set is at most $2 \log f_i + 3$.

Suppose that m of the end segments $F_1, \ldots, F_k$ have at most 3 faults. Without loss of generality we may assume that these are end segments $F_1, \ldots, F_m$. Procedure Fast-Line-Scan uses at most 4 tests for each of them. By definition we have $f_i \geq 4$ for all $m + 1 \leq i \leq k$. For these i, Procedure Fast-Line-Scan uses at most $2 \log f_i + 3 < 4 \log f_i$ tests. Hence the total number of tests is at most $4m + \sum_{i=m+1}^{k} 4 \log f_i$ which is at most $4m + 4(k-m) \log \frac{\lambda}{k-m}$, in view of Proposition 2. We have $k/(k-m) \geq 1$ and $\alpha > 31$, which implies

$$\log \alpha + \log \frac{k}{k-m} < \frac{\alpha - 4}{4} \cdot \frac{k}{k-m}.$$

It follows that

$$4(k-m) \log \frac{\alpha k}{k-m} < (\alpha - 4)k,$$

and consequently

$$4m + 4(k-m) \log \frac{\lambda}{k-m} \leq 4k + 4(k-m) \log \frac{\lambda}{k-m} =$$

$$4k + 4(k-m) \log \frac{\alpha k}{k-m} < \alpha k = \lambda.$$

This means that the total number of tests used in the calls of Procedure Fast-Line-Scan for $F_1, \ldots, F_k$ is strictly smaller than the upper bound λ on the number of faults remaining to be diagnosed. Consequently, at least one test is saved and the total number of tests is at most $n + t - 1$ in this case as well.

References

1. Beigel, R., Hurwood, W., Kahale, N.: Fault diagnosis in a flash. In: Proc. 36th Symp. on Found. of Comp. Sci., pp. 571–580 (1995)
2. Beigel, R., Kosaraju, S.R., Sullivan, G.F.: Locating faults in a constant number of testing rounds. In: Proc. 1st Ann. ACM Symp. Par. Alg. and Arch., pp. 189–198 (1989)
3. Beigel, R., Margulis, G., Spielman, D.A.: Fault diagnosis in a constant number of parallel testing rounds. In: Proc. 5th Ann. ACM Symp. Par. Alg. and Arch., pp. 21–29 (1993)

4. Bjorklund, A.: Optimal adaptive fault diagnosis of hypercubes. In: Halldórsson, M.M. (ed.) SWAT 2000. LNCS, vol. 1851, pp. 527–534. Springer, Heidelberg (2000)
5. Blecher, P.M.: On a logical problem. Disc. Math. 43, 107–110 (1983)
6. Dahbura, A.T.: System-level diagnosis: A perspective for the third decade, Concurrent Computation: Algorithms, Architectures, Technologies. Plenum Press, New York (1988)
7. Fujita, S., Araki, T.: Three-round adaptive diagnosis in binary n-cubes. In: Fleischer, R., Trippen, G. (eds.) ISAAC 2004. LNCS, vol. 3341, pp. 442–451. Springer, Heidelberg (2004)
8. Hakimi, S.L., Nakajima, K.: On adaptive system diagnosis. IEEE Transactions on Computers 33, 234–240 (1984)
9. Lee, S., Shin, K.G.: Probabilistic diagnosis of multiprocessor systems. ACM Computing Surveys 26, 121–139 (1994)
10. Nakajima, K.: A new approach to system diagnosis. In: Proc. 19th Allerton Conf. Commun. Contr. and Computing, pp. 697–706 (1981)
11. Nomura, K., Yamada, T., Ueno, S.: On adaptive fault diagnosis for multiprocessor systems. In: Eades, P., Takaoka, T. (eds.) ISAAC 2001. LNCS, vol. 2223, pp. 86–98. Springer, Heidelberg (2001)
12. Pelc, A., Upfal, E.: Reliable fault diagnosis with few tests, Combinatorics. Probability & Computing 7, 323–333 (1998)
13. Preparata, F., Metze, G., Chien, R.: On the connection assignment problem of diagnosable systems. IEEE Transactions on Electron. Computers 16, 848–854 (1967)
14. Schmeichel, E., Hakimi, S.L., Otsuka, M., Sullivan, G.: On minimizing testing rounds for fault identification. In: Proc. 18th Int. Symp. Fault-Tolerant Computing, pp. 266–271. IEEE Computer Society Press, Los Alamitos (1988)

Dynamic Structures for Top-k Queries on Uncertain Data

Jiang Chen[1,⋆] and Ke Yi[2,⋆⋆]

[1] Center for Computational Learning Systems, Columbia University
New York, NY 10115, USA
criver@cs.columbia.edu
[2] Department of Computer Science and Engineering, Hong Kong University of Science and Technology, Clear Water Bay, Kowloon, Hong Kong
yike@cse.ust.hk

Abstract. In an *uncertain data set* $\mathcal{S} = (S, p, f)$ where S is the *ground set* consisting of n elements, $p : S \to [0, 1]$ a probability function, and $f : S \to \mathbb{R}$ a score function, each element $i \in S$ with score $f(i)$ appears independently with probability $p(i)$. The top-k query on $\mathcal{S}$ asks for the set of k elements that has the maximum probability of appearing to be the k elements with the highest scores in a random instance of $\mathcal{S}$. Computing the top-k answer on a fixed $\mathcal{S}$ is known to be easy. In this paper, we consider the dynamic problem, that is, how to maintain the top-k query answer when $\mathcal{S}$ changes, including element insertion and deletions in the ground set S, changes in the probability function p and the score function f. We present a fully dynamic data structure that handles an update in $O(k \log k \log n)$ time, and answers a top-j query in $O(\log n + j)$ time for any $j \leq k$. The structure has $O(n)$ size and can be constructed in $O(n \log^2 k)$ time. As a building block of our dynamic structure, we present an algorithm for the *all-top-k* problem, that is, computing the top-j answers for all $j = 1, \ldots, k$, which may be of independent interest.

1 Introduction

Uncertain data naturally arises in a number of modern applications, e.g. imprecise measurement in mobile and sensor data [6], fuzzy duplicates in data warehouse [2], data integration [9], data cleaning [8,4], etc. These applications have called for a lot of research activities in modeling and querying uncertain data in recent years. An uncertain data model represents a probability distribution of all the possible instances of the data set. For example, in the basic uncertain data model [5,1], an *uncertain data set* $\mathcal{S} = (S, p)$ consists of a *ground set* of elements $S = \{1, \ldots, n\}$ and a probability function $p : S \to [0, 1]$. It is assumed that each element i appears independently with probability $p(i)$, i.e., the probability that $\mathcal{S}$ instantiates into $I \subseteq S$ is

$$\Pr[I \mid \mathcal{S}] = \prod_{i \in I} p(i) \prod_{i \in S \setminus I} (1 - p(i)).$$

⋆ Supported in part by a research contract from Consolidated Edison.
⋆⋆ Supported in part by Hong Kong Direct Allocation Grant (DAG07/08).

T. Tokuyama (Ed.): ISAAC 2007, LNCS 4835, pp. 427–438, 2007.

This basic model, in spite of its simplicity, has often been used to approximate the uncertain nature of the underlying data set. We will also adopt this model in this paper. In the following, we use $I \sim \mathcal{S}$ to denote that I is a random instance generated from $\mathcal{S}$.

Top-k queries are perhaps the most common type of queries in such applications, and have attracted much attention recently. However, all of the existing works can only handle one-time top-k computations [12,10,13]. When the underlying data changes, i.e., when the associated probabilities change, or elements are inserted or deleted, the algorithm has to recompute the answer to the query. This is often unacceptable due to the inherent dynamic nature of the uncertain data in many applications. For instance in data integration, the probability $p(i)$ represents the confidence of its existence, as more data becomes available from different sources, it is conceivable that the confidence levels might experience frequent changes. In this paper, we are interested in designing dynamic data structures that can be used to efficiently maintain the correct top-k answer as the uncertain data set undergoes a series of updates, including probability updates, element insertions and deletions.

Problem definition. There exist a few definitions for top-k queries in the literature. We adopt the following natural definition [12], which also requires a score function $f : S \to \mathbb{R}$.

Definition 1. *[12] Let $\mathcal{S} = (S, p, f)$ be an uncertain data set. For any $I \subseteq S$, let $\Psi_k(I)$ be the top-k elements in I according to the score function f; if $|I| < k$, define $\Psi_k(I) = \emptyset$. Let T be any set of k elements. The answer T^* to a top-k query on $\mathcal{S}$ is $T^* = \arg\max_T \Pr_{I\sim\mathcal{S}}[\Psi_k(I) = T] = \arg\max_T \sum_{\Psi_k(I)=T} \Pr[I \mid \mathcal{S}]$. Ties can be broken arbitrarily.*

In other words, T^* is the set of k elements that has the maximum probability of being at the top-k according to the score function in a randomly generated instance. As a concrete example, S can be a collection of sensors deployed in an environmental study, f represents their precipitation readings, and p measures the probabilities that the sensors are functioning normally. Thus, the top-k result gives us a good idea of where high precipitation occurs. Please see [12] for more potential applications.

As a convention, we assume that all the scores are distinct and $\mathcal{S}$ is given in the decreasing score order, i.e., $f(1) > f(2) > \cdots > f(n)$. Thus the probability of a set T of size k being the top-k elements $\Pr_{I\sim\mathcal{S}}[\Psi_k(I) = T]$ becomes

$$\prod_{j\in T} p(j) \prod_{j<l(T), j\notin T} (1 - p(j))$$

where $l(T)$ is the last element in T. The problem becomes finding the set of k elements T^* that maximizes the above quantity. In this paper, we break ties by choosing the T^* with a smaller $l(T^*)$.

Previous work. Quite a few uncertain data models have been proposed in the database literature [11,3,1,5]. They range from the basic model that we use in

this paper, to powerful models that are *complete*, i.e., models that can represent any probability distribution of the data set instances. However, complete models have exponential complexities and are hence uninteresting computationally. Some extensions to the basic model have been introduced to expand the expressiveness of the model while keeping computation tractable. Notably, in the TRIO [1] system, an uncertain data set consists of a number of *x-tuples*, and each x-tuple may include a number of elements associated with probabilities, and represent a discrete probability distribution of these elements being selected. Independence is still assumed among the x-tuples.

Soliman et al. [12] first proposed the problem of top-k query processing in an uncertain data set. Their algorithms have been recently improved by Yi et al. [13], both in the basic uncertain data model and the x-tuple model. In the basic model, if the elements are given in the sorted score order, there is a simple $O(n \log k)$-algorithm to compute the answer of the top-k query in one pass [13]. We scan the elements one by one, and maintain in a heap the k elements with the highest probabilities seen so far. Every time the heap changes we also incrementally compute the probability of these k elements being the top-k answer, i.e., the probability that all of these k elements appear multiplied by the probability that none of the other seen elements appear. In the end we report the k elements that achieve the maximum probability. However, this simple algorithm is inherently static, it is not clear how to extend it to handle updates without recomputation. As illustrated by the above sensor example, the uncertain data set may experience frequent changes, therefore it is important to develop dynamic algorithms for the problem.

There are a few other top-k query definitions proposed recently. For example, Soliman et al. [12] also proposed the U-kRanks query that concerns with the probability of an element appearing at a particular rank in a randomly generated instance. Another different framework by Ré et al. [10] deals with the problem of finding the k most probable answer for a given certain query, and there the additional scoring dimension is not involved.

Our results. In this paper, we present a dynamic structure of size $O(n)$ that always maintains the correct answer to the top-k query for an uncertain data set $\mathcal{S}$. In fact, we support more general queries than just for a specific k. Given any $j \leq k$, our structure answers the top-j query in time $O(\log n + j)$. We conjecture that the problem does not necessarily become easier even if one only requires support for the top-k query. Our structure takes $O(k \log k \log n)$ time to process a probability update, insert a new element into $\mathcal{S}$, or delete an element from $\mathcal{S}$. Note that a score change can be simply accomplished by an element deletion followed by an insertion. Given an uncertain data set whose elements are sorted by score, it takes $O(n \log^2 k)$ time to build the structure. The new structure uses a different approach than the static algorithm, and is based on a decomposition of the problem, which allows for efficient updates.

Before presenting our dynamic data structure, in Section 2 we consider a generalized version of the top-k problem, the so called *all-top-k* problem, in

which we want to compute the top-j answers for all $j = 1, \ldots, k$. We give an $O(n \log^2 k + k^2)$-time algorithm for this problem. This algorithm is also a building block of our dynamic data structure, which we describe in Section 3.

2 The All-Top-k Problem

In this section, we consider a slightly generalized version of the basic top-k problem. Given an uncertain data set $\mathcal{S}$, in the *all-top-k* problem, we want to compute the answers to all the top-j queries, for $j = 1, \ldots, k$. Naïvely applying the basic algorithm in [13] for each j would result in a total running time of $O(nk \log k)$. Below we give an $O(n \log^2 k + k^2)$ algorithm, which will also be useful in our dynamic structure presented in Section 3. Note that the k^2 term in the upper bound is necessary because this problem has a total result size of $\Theta(k^2)$.

Henceforth we will denote the top-j answer as T_j^*. We first observe that once we know $l(T_j^*)$, the last element in T_j^*, the other $j-1$ elements of T_j^* are simply the $j-1$ highest-probability elements in $[1, l(T_j^*)]$. In the following, we focus on computing $l(T_j^*)$ for all j, and present an algorithm that runs in $O(n \log^2 k)$ time. After we have the $l(T_j^*)$'s, the T_j^*'s can be computed easily in $O(n \log k + k^2)$ time by scanning all the elements again while keeping a heap of size k. If only the probabilities of these top-j answers are required, $O(n \log k)$ time suffices.

Algorithm outline. To simplify our notation, we use l_j (for $1 \le j \le k$) to denote $l(T_j^*)$, the last element in T_j^*. We will progressively process the first i elements of $\mathcal{S}$ and update the corresponding l_j's, as i goes from 1 to n. When we finish processing all n elements, we obtain the l_j's for $\mathcal{S}$. However, since there are $\Theta(nk)$ such values (k values for each position i), we cannot even afford to list all of them explicitly; instead, we store them in a "compressed list" that allows for fast updates. The data structure makes essential use of the following two properties of the changes these l_j's may experience. The first property is monotonicity. Note that the following lemma holds for all uncertainty data sets, including those consisting of the first i elements of $\mathcal{S}$.

Lemma 1. *For any $1 \le j < j' \le k$, we have that $l_j \le l_{j'}$.*

Proof. We only need to prove the case when $j' = j+1$, and the general statement will be an easy consequence. Let T be a set of size j, and $e \notin T$, denote by $r(T, e)$ the ratio of the probability of $T \cup \{e\}$ being the top-$(j+1)$ set to that of T being the top-j. We have[1]

[1] In this paper, we use the following convention to handle the multiplication and division of zeros. We keep a counter on how many zeroes have been applied to a product: incrementing the counter for each multiplication by 0 and decrementing for each division by 0. We interpret the final result as 0 if the counter is positive, or ∞ if negative.

$$r(T,e) = \frac{\mathrm{P}_{I\sim\mathcal{S}}\, T\cup\{e\} = \Psi_{j+1}(I)}{\mathrm{P}_{I\sim\mathcal{S}}\, T = \Psi_j(I)} = \frac{\prod_{h\in T\cup\{e\}} p(h)\prod_{h<l(T\cup\{e\}),h\notin T\cup\{e\}}(1-p(h))}{\prod_{h\in T} p(h)\prod_{h<l(T),h\notin T}(1-p(h))}$$
$$= \begin{cases} \frac{p(e)}{1-p(e)}, & e < l(T), \\ p(e)\prod_{l(T)<h<e}(1-p(h)), & e > l(T). \end{cases}$$

Note that when $e > l(T)$, $r(T,e) = \frac{p(e)}{1-p(e)}\prod_{l(T)<h\le e}(1-p(h)) \le \frac{p(e)}{1-p(e)}$.

Now assuming on the contrary $l_j > l_{j+1}$, let e be an element in T^*_{j+1} but not in T^*_j. We will show that $T^*_j \cup \{e\}$ is more likely to be the top-$(j+1)$ set than T^*_{j+1}, which leads to contradiction. Since $e < l_j$, $r(T^*_j, e) = \frac{p(e)}{1-p(e)} \ge r(T,e)$ for any T. Because $l_j > l_{j+1}$, by the definition of T^*_j and our tie breaking rule, we must have $\Pr_{I\sim\mathcal{S}}[T^*_j = \Psi_j(I)] > \Pr_{I\sim\mathcal{S}}[T^*_{j+1}\setminus\{e\} = \Psi_j(I)]$. Therefore, the probability that $T^*_j \cup \{e\}$ is the top-$(j+1)$ answer is

$$\begin{aligned} \Pr_{I\sim\mathcal{S}}[T^*_j = \Psi_j(I)]\cdot r(T^*_j, e) &> \Pr_{I\sim\mathcal{S}}[T^*_{j+1}\setminus\{e\} = \Psi_j(I)]\cdot r(T^*_j, e) \\ &\ge \Pr_{I\sim\mathcal{S}}[T^*_{j+1}\setminus\{e\} = \Psi_j(I)]\cdot r(T^*_{j+1}\setminus\{e\}, e) \\ &= \Pr_{I\sim\mathcal{S}}[T^*_{j+1} = \Psi_{j+1}(I)], \end{aligned}$$

a contradiction.

The second property is that, when we process the i-th element, l_j either changes to i or stays the same because all newly added j-sets contains i. By Lemma 1, if l_j changes to i, so do all $l_{j'}$'s for $j \le j' \le k$. Thus, to process element i, the problem basically becomes finding the smallest j such that l_j becomes i.

Updating the l_j's. We store $l_1, \ldots, l_{\min\{i,k\}}$ in a list, both j and the value of l_j. By Lemma 1 this list is automatically in the increasing order of both j and l_j. We further compress the list by representing the l_j's with equal values by ranges. For example, if $l_1 = 1, l_2 = l_3 = l_4 = 5, l_5 = l_6 = 6$, then the list looks like $(1,[1,1]),(5,[2,4]),(6,[5,6])$. Suppose that we have a comparison method to decide if l_j becomes i for any j, then we can locate the minimum such j, denoted j^*, as follows. We first visit the compressed list from right to left, checking the boundaries of each range, until we locate the range that contains j^*. Next we do a binary search inside the range to pin down its exact location. Finally, supposing that the entry in the list whose range contains j^* is $(i',[j_1,j_2])$, we first truncate all the trailing entries in the list, and then replace $(i',[j_1,j_2])$ with $(i',[j_1,j^*-1])$ (if $j_1 \le j^*$) and $(i,[j^*,i])$. Note that a special case is when j^* does not exist, i.e., no l_j becomes i. In this case if $i \le k$, we append $(i,[i,i])$ to the list; otherwise we do nothing.

We bound the number of comparisons per element as follows. In the first step when we scan the list from right to left, if we pass an entry, then it will be removed immediately. Thus, the amortized number of comparisons is $O(1)$ for the first step. The second step involves a binary search inside a range of length at most k, which needs $O(\log k)$ comparisons. Therefore, the algorithm performs $O(n\log k)$ comparisons for all n elements.

The comparison method. To complete the algorithm, we finally specify how to conduct each comparison in the algorithm above, which decides whether some l_j should change to i. Let $T_j^*(l)$ be the highest-probability j-set whose last element is l, i.e., $T_j^*(l)$ consists of l and the $j-1$ elements in $\{1,\ldots,l-1\}$ with the largest probabilities. We need to compute both $\Pr_{I\sim\mathcal{S}}[\Psi_j(I) = T_j^*(l_j)]$ and $\Pr_{I\sim\mathcal{S}}[\Psi_j(I) = T_j^*(i)]$ and compare them. Recall that for a set T of size j,

$$\begin{aligned}\Pr_{I\sim S}[\Psi_j(I) = T] &= \prod_{e\in T} p(e) \prod_{e<l(T),e\notin T} (1-p(e)) \\ &= \prod_{e\in T} \frac{p(e)}{1-p(e)} \prod_{e\le l(T)} (1-p(e)).\end{aligned}$$

The second factor is simply a prefix-product and can be easily maintained for all $l(T)$ with a table of size $O(n)$. To compute $\prod_{e\in T} \frac{p(e)}{1-p(e)}$ for $T = T_j^*(l_j)$ and $T = T_j^*(i)$, we build a data structure that supports the following queries: given any j, l, return the product of $p(e)/(1-p(e))$'s for the $j-1$ highest-probability elements e in $\{1,\ldots,l-1\}$. Below we give such a structure, which answers a query in $O(\log k)$ time and can be constructed in $O(n\log k)$ time. It is obvious that with this structure, we can perform a comparison in $O(\log k)$ time, leading to a total running time of $O(n\log^2 k)$ to process all n elements.

Again we process the n elements one by one, and maintain a dynamic binary tree (say a red-black tree) of k elements, storing the highest-probability elements among the elements that have been processed, sorted by their probabilities. At the leaf of the tree storing e, we maintain the value $p(e)/(1-p(e))$, and in each internal node u the product of all $p(e)/(1-p(e))$'s in the subtree rooted at u. It can be verified that this binary tree can be updated in $O(\log k)$ time per element. The binary tree built after having processed the first $l_j - 1$ elements can be used to compute $\prod_{e\in T} \frac{p(e)}{1-p(e)}$ for $T = T_j^*(l_j)$ in $O(\log k)$ time. The same can be said for i and $T_j^*(i)$. However, the comparison of $T_j^*(l_j)$ and $T_j^*(i)$ requires queries on both binary trees, which are not supported by the progressive processing.

To support queries for all binary trees that ever appear, we make the data structure *partially persistent*, i.e., the structure has multiple versions, one corresponding to each binary tree ever built, and allows queries on any version, but only allows updates to the current version. That is, when we process i, we produce a new binary tree of version i without altering any of the previous versions. Since the binary tree clearly has bounded in-degree, we can use the generic technique of Driscoll et al. [7] to make it partially persistent, without increasing the asymptotic query and update costs. Thus, this persistent structure can be built in $O(n\log k)$ time and supports a query on any version of the binary tree in time $O(\log k)$. However, the space requirement increases to $O(n\log k)$.

This completes the description of the algorithm.

Theorem 1. *There is an algorithm that computes $l_1,\ldots,l_k$ in $O(n\log^2 k)$ time.*

As described at the beginning of this section, this leads to the following corollary.

Corollary 1. *There is an algorithm that solves the all-top-k problem in* $O(n \log^2 k + k^2)$ *time.*

3 The Dynamic Data Structure

We present our dynamic data structure in this section. We begin with probability updates, and assume that the ground set S is static. In Section 3.1, we describe our data structure, which can be updated in time with a naïve $O(k^2 \log n)$ algorithm. We then present a better node merging algorithm in Section 3.2, improving the update time to $O(k \log k \log n)$. Finally, in Section 3.3, we talk about how to handle element insertions and deletions.

3.1 The Data Structure

The structure. We build a balanced binary tree $\mathcal{T}$ on $\{1, \ldots, n\}$. Each leaf of $\mathcal{T}$ stores between k and $2k$ elements. Thus there are a total of $O(n/k)$ leaves, and hence a total number of $O(n/k)$ nodes in $\mathcal{T}$. For any node $u \in \mathcal{T}$, let S^u be the set of elements stored in the leaves of the subtree rooted at u, and $\mathcal{S}^u$ be the corresponding uncertain data set.

For each node u, we solve the all-top-k problem for $\mathcal{S}^u$, except that we do not list or store the all-top-k sets (which takes time and space of $\Omega(k^2)$). Instead, we only store the corresponding probabilities of the sets. More precisely, let $T_j^*(\mathcal{S}^u)$ be the top-j answer for $\mathcal{S}^u$. We compute and store $\rho_j^u = \Pr_{I \sim \mathcal{S}^u}[\Psi_j(I) = T_j^*(\mathcal{S}^u)]$ for all $j = 1, \ldots, k$. Thus the all-top-k solutions for the whole set $\mathcal{S}$ can be found at the root of the whole binary tree.

At each node u, we also compute $k+1$ auxiliary variables π_j^u, for $j = 0 \ldots, k$. If we sort the elements in $\mathcal{S}^u$ by their probabilities in descending order, and suppose that $e_1^u, e_2^u, \ldots, e_{|S^u|}^u$ is such an order, then π_j^u is defined as

$$\pi_j^u = \prod_{h=1}^{j} p(e_h^u) \prod_{h=j+1}^{|S^u|} (1 - p(e_h^u)). \tag{1}$$

In other words, π_j^u is the maximum probability for any j-set generated from $\mathcal{S}^u$. Note that $\pi_0^u = \prod_{e \in S^u}(1 - p(e))$ is just the probability that none of S^u appears.

This completes the description of our data structure. It is obvious that the structure has a size of $O(n)$.

Initializing and updating the π_j^u*'s.* Rewriting (1), we get

$$\pi_j^u = \prod_{h=1}^{j} \frac{p(e_h^u)}{1 - p(e_h^u)} \prod_{h=1}^{|S^u|} (1 - p(e_h^u)) = \pi_0^u \cdot \prod_{h=1}^{j} \frac{p(e_h^u)}{1 - p(e_h^u)}. \tag{2}$$

Hence π_j^u is just the j-th prefix product of the list $\frac{p(e_1^u)}{1-p(e_1^u)}, \frac{p(e_2^u)}{1-p(e_2^u)}, \ldots$ times π_0^u. This suggests us to maintain the list up to the first k elements. These can be

prepared for all the leaves u in $O(n \log k)$ time by sorting the elements in each S^u by their probabilities. For an internal node u with children v and w, we just merge the two lists associated with v and w, which takes $O(k)$ time. To compute π_0^u takes $O(k)$ time per leave, but only $O(1)$ time for an internal node because $\pi_0^u = \pi_0^v \pi_0^w$. Given the list and π_0^u, all π_j^u's can be computed in time $O(k)$ for u. Thus it takes time $O(n \log k)$ to initialize all the π_j^u's. When there is probability change at a leaf, we can update all the affected π_j^u's in $O(k \log n)$ time along the leaf-to-root path.

Initializing and updating the ρ_j^u's. Now we proceed to the more difficult part, maintaining the ρ_i^u's. For a leaf e, the ρ_j^e's can be computed by invoking the algorithm in Section 2, taking $O(k \log^2 k)$ time per leaf and $O(n \log^2 k)$ overall. For an internal node u, ρ_i^u can be computed as specified in the following lemma.

Lemma 2. *Let u be an internal node with v and w being its left and right child, respectively. For any $1 \leq j \leq k$,*

$$\rho_j^u = \max\{\rho_j^v, \max_{1 \leq h \leq j} \pi_{j-h}^v \rho_h^w\}. \tag{3}$$

Proof. Recall that the leaves of the tree are sorted in the descending order of score. Thus the left child of u, namely v, contains elements with higher scores.

By definition, ρ_j^u is the top-j query answer for the uncertain data set $\mathcal{S}^u$. There are two cases for the top-j query answer. Either we choose all of these j elements from S^v, which has a maximum probability of ρ_j^v, or choose at least one element from S^w. The latter case is further divided into j sub-cases: We can choose $j - h$ elements from S^v and h elements from S^w, for $h = 1, \ldots, j$. For each sub-case, the maximum probability is $\pi_{j-h}^v \rho_h^w$.

The naïve way to maintain the ρ_j^u's is to compute (3) straightforwardly, which takes $\Theta(k^2)$ time per internal node. In Section 3.2 we present an improved node merging algorithm that computes all ρ_j^u's for u in time $O(k \log k)$. This will lead to an overall initialization time of $(n \log^2 k)$ for the whole structure, and an update time of $O(k \log k \log n)$.

Querying the structure. Once we have the structure available, we can easily extract the top-k query answer by remembering which choice we have made for each ρ_j^u in Lemma 2. We briefly outline the extraction algorithm here. We visit $\mathcal{T}$ in a top-down fashion recursively, starting at the root querying for its top-k answer. Suppose we are at node $u \in \mathcal{T}$ with children v and w, querying for its top-j answer. If $\rho_j^u = \rho_j^v$, then we recursively query v for its top-j answer. Otherwise, suppose $\rho_j^u = \pi_{j-h}^v \rho_h^w$ for some h. We report $e_1^v, \ldots, e_{j-h}^v$ and then recursively query w for its top-h answer. It is not difficult to see that this extraction process takes $O(\log n + k)$ time in total.

Note that our data structure is capable of answering queries for any top-j, $j \leq k$. It is not clear to us whether restricting to only the top-k answer will make the problem any easier. We suspect that the all-top-k feature of our

data structure is inherent in the problem of maintaining only the top-k answer. For example, in the case when the probability of the element with the highest score, namely $p(1)$, is 0, we need to compute the top-k answer of the rest $n-1$ elements. However, when $p(1)$ is changed to 1, the top-k answer changes to $\{1\}$ union the top-$(k-1)$ answer of the rest of $n-1$ elements. This example can be further generalized. When $p(1), p(2), \ldots, p(k-1)$ are changed from 0 to 1 one after another, the top-k answer of the whole data set is changed from the top-k answer, to the top-$(k-1)$ answer, then to the top-$(k-2)$ answer,..., and finally to the top-1 answer of the rest $n-k+1$ elements.

3.2 An Improved Node Merging Algorithm

Naïvely evaluating (3) takes $\Theta(k^2)$ time. In this section, we present an improved $O(k \log k)$-time algorithm. In the following, we concentrate on computing the second terms inside the max of (3), with which computing ρ_j^u's takes only k max-operations. That is, we focus on computing $\bar{\rho}_j^u = \max_{1 \le h \le j} \pi_{j-h}^v \rho_h^w$, for $j = 1, \ldots, k$.

Our algorithm exploits the internal structure of the problem. In Figure 1, we represent each product $\pi_{j-h}^v \rho_h^w$ by a square. Thus, each $\bar{\rho}_j^u$ is the maximum over the corresponding diagonal. We number the diagonals from top-left to bottom-right, so that the product $\pi_{j-h}^v \rho_h^w$ is in diagonal j. We will again make use of the monotonicity property similar to that shown in Lemma 1. For two columns h and h' of Figure 1, we say column h *beats* column h' at diagonal j, where $2 \le h \le j$, if the product at the intersection of column h and diagonal j is larger than that at the intersection of column h' and diagonal j. The following lemma shows that these comparisons between two columns exhibit monotonicity.

Lemma 3. *For any $1 \le h' < h \le j$, if column h beats column h' at diagonal j, then column h beats column h' at any diagonal j', where $j \le j' \le k$.*

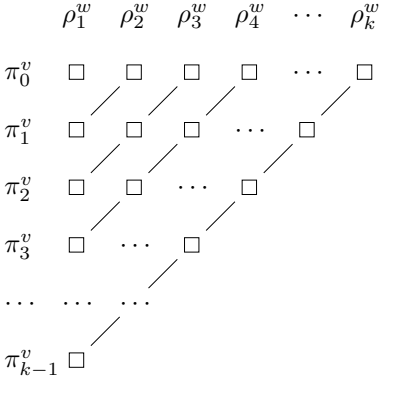


Fig. 1. A square represents the product of the corresponding π_{j-h}^v and ρ_h^w. Each $\bar{\rho}_j^u$ is the maximum on a diagonal.

Proof. By assumption, we have

$$\pi^v_{j-h}\rho^w_h > \pi^v_{j-h'}\rho^w_{h'}. \tag{4}$$

Rewriting (4), we get

$$\frac{\rho^w_h}{\rho^w_{h'}} > \frac{\pi^v_{j-h'}}{\pi^v_{j-h}}. \tag{5}$$

Recalling (2), the RHS of (5) can be expressed as

$$\prod_{t=j-h+1}^{j-h'} \frac{p(e^v_t)}{1-p(e^v_t)},$$

which becomes smaller when j becomes larger (remember that the e^v_t's are sorted by probabilities in descending order). Therefore, (5), and hence (4), will also hold if we replace j with j'.

We will progress column by column, and for each diagonal keep the current "winner", i.e., the column that beats all the other columns seen so far. After we have processed column j (the last column that has intersection with diagonal j), the winner for diagonal j then determines $\bar{\rho}^u_j$, and we can remove diagonal j from the current maintained list.

By Lemma 3, the way how the winners change exhibits the same pattern as the l_j's do in Section 2. More precisely, when we process column j, if h^* is the minimum h such that the winner of diagonal h changes to column j, then all diagonals after h^* will have their winners changed to j. Thus, we can use the same algorithm (using a compressed list) that we designed for computing the l_j's in Section 2 to maintain the list of winners. Since here comparing two columns at a particular diagonal takes $O(1)$ time (as opposed to $O(\log k)$ in Section 2), the total running time is $O(k \log k)$.

Therefore, we can compute the ρ^u_j's in $O(k \log k)$ time for each node u. To summarize, when the probability of an element changes, we first update all the π^u_j values for all the nodes on a leaf-to-root path, taking $O(k)$ time per node. Next, we recompute the ρ^u_j values at the leaf containing the updated element. This takes $O(k \log^2 k)$ time using our all-top-k algorithm of Section 2. Finally, we update the other ρ^u_j values for all nodes on the leaf-to-root path in a bottom-up fashion, taking $O(k \log k)$ time per node. The overall update cost is thus $O(k \log^2 k + k \log k \log n) = O(k \log k \log n)$.

3.3 Handling Element Insertions and Deletions

We can handle element insertions and deletions using standard techniques. We make the binary tree $\mathcal{T}$ a dynamic balanced binary tree, say a red-black tree, sorted by scores. To insert a new element, we first find the leaf where the element should be inserted. If the leaf contains less than $2k$ elements, we simply insert the new element, and then update all the affected π^u_i and ρ^u_i values as described

previously. If the leaf already contains $2k$ elements, we split it into two, creating a new internal node, which becomes the parent of the two new leaves. After inserting the new element into one of the two new leaves, we update the π_i^u and ρ_i^u values as before. When the tree gets out of balance, we apply rotations. Each rotation may require the recomputation of the π_i^u and ρ_i^u values at a constant number of nodes, but this does not change the overall asymptotic complexity. Deletions can be handled similarly.

Therefore, we reach the main result of this paper.

Theorem 2. *There is a fully dynamic data structure that maintains an uncertain data set under probability changes, element insertions and deletions that takes $O(k \log k \log n)$ time per update, and answers a top-j query in $O(\log n + j)$ time for any $j \leq k$. The structure has size $O(n)$ and can be constructed in $O(n \log^2 k)$ time. All bounds are worst-case.*

4 Concluding Remarks

In this paper we present a dynamic data structure for the top-k problem with an update cost of $O(k \log k \log n)$. We conjecture that there is an inherent $\Omega(k)$ lower bound for the problem. As a building block of our main result, we also present an all-top-k algorithm that runs in $O(n \log^2 k + k^2)$ time.

Many directions for this problem remain elusive. For example, we have only considered the basic uncertain data model. It would be interesting if we can extend our approach to other more powerful models, such as the x-tuple model [1]. Another orthogonal direction is to consider other top-k definitions [12,10].

References

1. Agrawal, P., Benjelloun, O., Das Sarma, A., Hayworth, C., Nabar, S., Sugihara, T., Widom, J.: Trio: A system for data, uncertainty, and lineage. In: VLDB (2006)
2. Ananthakrishna, R., Chaudhuri, S., Ganti, V.: Eliminating fuzzy duplicates in data warehouses. In: VLDB (2002)
3. Benjelloun, O., Sarma, A.D., Halevy, A., Widom, J.: ULDBs: Databases with uncertainty and lineage. In: VLDB (2006)
4. Chaudhuri, S., Ganjam, K., Ganti, V., Motwani, R.: Robust and efficient fuzzy match for online data cleaning. In: SIGMOD (2003)
5. Dalvi, N., Suciu, D.: Efficient query evaluation on probabilistic databases. In: VLDB (2004)
6. Deshpande, A., Guestrin, C., Madden, S., Hellerstein, J., Hong, W.: Model-driven data acquisition in sensor networks. In: VLDB (2004)
7. Driscoll, J.R., Sarnak, N., Sleator, D.D., Tarjan, R.E.: Making data structures persistent. Journal of Computer and System Sciences 38(1), 86–124 (1989)
8. Galhardas, H., Florescu, D., Shasha, D.: Declarative data cleaning: Language, model, and algorithms. In: VLDB (2001)
9. Halevy, A., Rajaraman, A., Ordille, J.: Data integration: the teenage year. In: VLDB (2006)

10. Ré, C., Dalvi, N., Suciu, D.: Efficient top-k query evaluation on probalistic databases. In: ICDE (2007)
11. Sarma, A.D., Benjelloun, O., Halevy, A., Widom, J.: Working models for uncertain data. In: ICDE (2006)
12. Soliman, M.A., Ilyas, I.F., Chang, K.C.: Top-k query processing in uncertain databases. In: ICDE (2007)
13. Yi, K., Li, F., Srivastava, D., Kollios, G.: Improved top-k query processing in uncertain databases. Technical report, AT&T Labs, Inc. (2007)

Separating Populations with Wide Data: A Spectral Analysis

Avrim Blum ⋆, Amin Coja-Oghlan⋆⋆, Alan Frieze⋆⋆⋆, and Shuheng Zhou†

Carnegie Mellon University, Pittsburgh, PA 15213, USA

Abstract. In this paper, we consider the problem of partitioning a small data sample drawn from a mixture of k product distributions. We are interested in the case that individual features are of low average quality γ, and we want to use as few of them as possible to correctly partition the sample. We analyze a spectral technique that is able to approximately optimize the total data size—the product of number of data points n and the number of features K—needed to correctly perform this partitioning as a function of $1/\gamma$ for $K > n$. Our goal is motivated by an application in clustering individuals according to their population of origin using markers, when the divergence between any two of the populations is small.

1 Introduction

We explore a type of classification problem that arises in the context of computational biology. The problem is that we are given a small sample of size n, e.g., DNA of n individuals (think of n in the hundreds or thousands), each described by the values of K *features* or *markers*, e.g., SNPs (Single Nucleotide Polymorphisms, think of K as an order of magnitude larger than n). Our goal is to use these features to classify the individuals according to their population of origin. Features have slightly different probabilities depending on which population the individual belongs to, and are assumed to be independent of each other (i.e., our data is a small sample from a mixture of k very similar product distributions). The objective we consider is to minimize the total data size $D = nK$ needed to correctly classify the individuals in the sample as a function of the "average quality" γ of the features, under the assumption that $K > n$. Throughout the paper, we use p_i^j and μ_i^j as shorthands for $p_i^{(j)}$ and $\mu_i^{(j)}$ respectively.

Statistical Model: We have k probability spaces $\Omega_1, \ldots, \Omega_k$ over the set $\{0,1\}^K$. Further, the components (*features*) of $z \in \Omega_t$ are independent and $\mathbf{Pr}_{\Omega_t}[z_i = 1] = p_t^i$ $(1 \le t \le k,\ 1 \le i \le K)$. Hence, the probability spaces $\Omega_1, \ldots, \Omega_k$ comprise the distribution of the features for each of the k populations. Moreover, the input of the algorithm consists of a collection (*mixture*) of $n = \sum_{t=1}^{k} N_t$ unlabeled samples, N_t points from Ω_t, and the algorithm is to determine for each data point from which of $\Omega_1, \ldots, \Omega_k$ it was chosen. In general we do *not* assume that $N_1, \ldots, N_t$ are revealed

⋆ Supported in part by the NSF under grant CCF-0514922.
⋆⋆ Supported by the German Research Foundation under grant CO 646.
⋆⋆⋆ Supported in part by the NSF under grant CCF-0502793.
† Supported in part by the NSF under grants CCF-0625879 and CNF–0435382.

T. Tokuyama (Ed.): ISAAC 2007, LNCS 4835, pp. 439–451, 2007.

to the algorithm; but we do require some bounds on their relative sizes. An important parameter of the probability ensemble $\Omega_1, \ldots, \Omega_k$ is the *measure of divergence*

$$\gamma = \min_{1 \le s < t \le k} \frac{\sum_{i=1}^{K} (p_s^i - p_t^i)^2}{K} \tag{1}$$

between any two distributions. Note that $\sqrt{K\gamma}$ measures the Euclidean distance between the means of any two distributions and thus represents their separation. Further, let $N = n/k$ (so if the populations were balanced we would have N of each type) and assume from now on that $kN < K$. Let $D = nK$ denote the size of the data-set. In addition, let $\sigma^2 = \max_{i,t} p_t^i(1 - p_t^i)$ denote the maximum variance of any random bit.

The biological context for this problem is we are given DNA information from n individuals from k populations of origin and we wish to classify each individual into the correct category. DNA contains a series of markers called SNPs, each of which has two variants (alleles). Given the population of origin of an individual, the genotypes can be reasonably assumed to be generated by drawing alleles independently from the appropriate distribution. The following theorem gives a sufficient condition for a balanced ($N_1 = N_2$) input instance when $k = 2$.

Theorem 1. (Zhou 06 [25]) *Assume $N_1 = N_2 = N$. If $K = \Omega(\frac{\ln N}{\gamma})$ and $KN = \Omega(\frac{\ln N \log\log N}{\gamma^2})$ then with probability $1 - 1/\operatorname{poly}(N)$, among all balanced cuts in the complete graph formed among $2N$ sample individuals, the maximum weight cut corresponds to the partition of the $2N$ individuals according to their population of origin. Here the weight of a cut is the sum of weights across all edges in the cut, and the edge weight equals the Hamming distance between the bit vectors of the two endpoints.*

Variants of the above theorem, based on a model that allows two random draws from each SNP for an individual, are given in [3,25]. In particular, notice that edge weights based on the inner-product of two individuals' bit vectors correspond to the sample covariance, in which case the max-cut corresponds to the correct partition [25] with high probability. Finding a max-cut is computationally intractable; hence in the same paper [3], a hill-climbing algorithm is given to find the correct partition for balanced input instances but with a stronger requirement on the sizes of both K and nK.

A Spectral Approach: In this paper, we construct two simpler algorithms using spectral techniques, attempting to reproduce conditions above. In particular, we study the requirements on the parameters of the model (namely, γ, N, k, and K) that allow us to classify every individual correctly and efficiently with high probability.

The two algorithms CLASSIFY and PARTITION compare as follows. Both algorithms are based on spectral methods originally developed in graph partitioning. More precisely, Theorem 2 is based on computing the singular vectors with the two largest singular values for each of the $n \times K$ input random matrix. The procedure is conceptually simple, easy to implement, and efficient in practice. For simplicity, Procedure Classify assumes the separation parameter γ is known to decide which singular vector to examine; in practice, one can just try both singular vectors as we do in the simulations. Proof techniques for Theorem 2, however, are difficult to apply to cases of multiple populations, i.e., $k > 2$. Procedure Partition is based on computing a rank-k approximation of

the input random matrix and can cope with a mixture of a constant number of populations. It is more intricate for both implementation and execution than Classify. It does not require γ as an input, while only requires that the constant k is given. We prove the following theorems.

Theorem 2. *Let* $\omega = \frac{\min(N_1,N_2)}{n}$ *and* $\omega_{\min}$ *be a lower bound on* ω. *Let* γ *be given. Assume that* $K > 2n \ln n$ *and* $k = 2$. *Procedure* CLASSIFY *allows us to separate two populations w.h.p., when* $n \geq \Omega\left(\frac{\sigma^2}{\gamma\omega_{\min}\omega}\right)$, *where* σ^2 *is the largest variance of any random bit, i.e.* $\sigma^2 = \max_{i,t} p_t^i(1-p_t^i)$. *Thus if the populations are roughly balanced, then* $n \geq \frac{c}{\gamma}$ *suffices for some constant* c.

This implies that the data required is $D = nK = O\left(\ln n\sigma^4/\gamma^2\omega^2\omega_{\min}^2\right)$. Let $P_s = (p_s^i)_{i=1,\ldots,K}$, we have

$$\|P_1 - P_2\|_2 = \sqrt{K\gamma} = \sqrt{\sum_{i=1}^{K}(p_1^i - p_2^i)^2} \geq \frac{\sigma}{\omega_{\min}\omega}\sqrt{\ln n}. \tag{2}$$

Theorem 3. *Let* $\omega = \frac{\min(N_1,\ldots,N_k)}{n}$. *There is a polynomial time algorithm* PARTITION *that satisfies the following. Suppose that* $K > n \log n$ *and* $n > \frac{C_k\sigma^2}{\gamma\omega}$ *for some large enough constant* C_k *and that* $\omega = \Omega(1)$. *Then given the empirical* $n \times K$ *matrix comprising the* K *features for each of the* n *individuals along with the parameter* k, PARTITION *separates the* k *populations correctly w.h.p.*

Summary and Future Direction: Note that unlike Theorem 1, both Theorem 2 and Theorem 3 require a lower bound on n, even when $k = 2$ and the input instance is balanced. We illustrate through simulations to show that this seems not to be a fundamental constraint of the spectral techniques; our experimental results show that even when n is small, by increasing K so that $nK = \Omega(1/\gamma^2)$, one can classify a mixture of two populations using ideas in Procedure Classify with success rate reaching an "oracle" curve, which is computed assuming that distributions are known, where success rate means the ratio between correctly classified individuals and N. Exploring the tradeoffs of n and K that are sufficient for classification, when sample size n is small, is both of theoretical interests and practical value.

1.1 Related Work

In their seminal paper [21], Pritchard, Stephens, and Donnelly presented a model-based clustering method to separate populations using genotype data. They assume that observations from each cluster are random from some parametric model. Inference for the parameters corresponding to each population is done jointly with inference for the cluster membership of each individual, and k in the mixture, using Bayesian methods.

The idea of exploiting the eigenvectors with the first two eigenvalues of the adjacency matrix to partition graphs goes back to the work of Fiedler [12], and has been used in the heuristics for various NP-hard graph partitioning problems (e.g., [13]). The main difference between graph partitioning problems and the classification problem

that we study is that the matrices occurring in graph partitioning are symmetric and hence diagonalizable, while our input matrix is rectangular in general. Thus, the contribution of Theorem 2 is to show that a conceptually simple and efficient algorithm based on singular value decompositions performs well in the framework of a fairly general probabilistic model, where probabilities for each of the K features for each of the k populations are allowed to vary. Indeed, the analysis of CLASSIFY requires exploring new ideas such as the Separation Lemma and the normalization of the random matrix X, for generating a large gap between top two singular values of the expectation matrix $\mathcal{X}$ and for bounding the angle between random singular vectors and their static correspondents, details of which are included in Section 2 with analysis in full version.

Procedure Partition and its analysis build upon the spectral techniques of McSherry [18] on graph partitioning, and an extension due to Coja-Oghlan [4]. McSherry provides a comprehensive probabilistic model and presents a spectral algorithm for solving the partitioning problem on random graphs, provided that a separation condition similar to (2) is satisfied. Indeed, [18] encompasses a considerable portion of the prior work on Graph Coloring, Minimum Bisection, and finding Maximum Clique. Moreover, McSherry's approach easily yields an algorithm that solves the classification problem studied in the present paper under similar assumptions as in Theorem 3, provided that the algorithm is given the parameter γ as an additional input; this is actually pointed out in the conclusions of [18]. In the context of graph partitioning, an algorithm that does not need the separation parameter as an input was devised in [4]. The main difference between PARTITION and the algorithm presented in [4] is that PARTITION deals with the asymmetric $n \times K$ matrix of individuals/features, whereas [4] deals with graph partitioning (i.e., a symmetric matrix).

There are two streams of related work in the learning community. The first stream is the recent progress in learning from the point of view of clustering: given samples drawn from a mixture of well-separated Gaussians (component distributions), one aims to classify each sample according to which component distribution it comes from, as studied in [8,9,2,23,1,15,7]. This framework has been extended to more general distributions such as log-concave distributions in [1,15] and heavy-tailed distributions in [7], as well as to more than two populations. These results focus mainly on reducing the requirement on the separations between any two centers P_1 and P_2. In contrast, we focus on the sample size D. This is motivated by previous results [3,25] stating that by acquiring enough attributes along the same set of dimensions from each component distribution, with high probability, we can correctly classify every individual.

While our aim is different from those results, where $n > K$ is almost universal and we focus on cases $K > n$, we do have one common axis for comparison, the ℓ_2-distance between any two centers of the distributions. In earlier works [9,2], the separation requirement depended on the number of dimensions of each distribution; this has recently been reduced to be independent of K, the dimensionality of the distribution for certain classes of distributions [1,15]. This is comparable to our requirement in (2) for the discrete distributions. For example, according to Theorem 7 in [1], in order to separate the mixture of two Gaussians,

$$\|P_1 - P_2\|_2 = \Omega\left(\frac{\sigma}{\sqrt{\omega}} + \sigma\sqrt{\log n}\right) \tag{3}$$

is required. Besides Gaussian and Logconcave, a general theorem: Theorem 6 in [1] is derived that in principle also applies to mixtures of discrete distributions. The key difficulty of applying their theorem directly to our scenario is that it relies on a concentration property of the distribution (Eq. (10) of [1]) that need not hold in our case. In addition, once the distance between any two centers is fixed (i.e., once γ is fixed in the discrete distribution), the sample size n in their algorithms is always larger than $\Omega\left(\frac{K}{\omega}\log^5 K\right)$ [1,15] for log-concave distributions (in fact, in Theorem 3 of [15], they discard at least this many individuals in order to correctly classify the rest in the sample), and larger than $\Omega(\frac{K}{\omega})$ for Gaussians [1], whereas in our case, $n < K$ always holds. Hence, our analysis allows one to obtain a clean bound on n in the discrete case.

The second stream of work is under the PAC-learning framework, where given a sample generated from some target distribution Z, the goal is to output a distribution Z_1 that is close to Z in Kullback-Leibler divergence: $KL(Z||Z_1)$, where Z is a mixture of product distributions over discrete domains or Gaussians [16,14,5,6,20,10,11]. They do not require a minimal distance between any two distributions, but they do not aim to classify every sample point correctly either, and in general require much more data.

2 A Simple Algorithm Using Singular Vectors

As described in Theorem 2, we assume we have a mixture of two product distributions. Let N_1, N_2 be the number of individuals from each population class. Our goal is to correctly classify all individuals according to their distributions. Let $n = 2N = N_1 + N_2$, and refer to the case when $N_1 = N_2$ as the balanced input case. For convenience, let us redefine "K" to assume we have $O(\log n)$ blocks of K features each (so the total number of features is really $O(K \log n)$) and we assume that each set of K features has divergence at least γ. (If we perform this partitioning of features into blocks randomly, then with high probability this divergence has changed by only a constant factor for most blocks.) The high-level idea of the algorithm is now to repeat the following procedure for each block of K features: use the K features to create an $n \times K$ matrix X, such that each row $X_i, i = 1, \ldots, n$, corresponds to a feature vector for one sample point, across its K dimensions. We then compute the top two left singular vectors u_1, u_2 of X and use these to classify each sample. This classification induces some probability of error f for each individual at each round, so we repeat the procedure for each of the $O(\log n)$ blocks and then take majority vote over different runs. Each round we require $K \geq n$ features, so we need $O(n \log n)$ features total in the end.

In more detail, we repeat the following procedure $O(\log n)$ times. Let $T = \frac{15N}{32}\sqrt{3\omega_{\min}\gamma}$, where $\omega_{\min}$ is the lower bound on the minimum weight $\min\{\frac{N_1}{2N}, \frac{N_2}{2N}\}$, which is independent of an actual instance. Let $s_1(X), s_2(X)$ be the top two singular values of X.

Procedure Classify: Given $\gamma, N, \omega_{\min}$. Assume that $N \gg \frac{1}{\gamma}$,

– Normalization: use the K features to form a random $n \times K$ matrix X; Each individual random variable $X_{i,j}$ is a *normalized* random variable based on the original Bernoulli r.v. $b_{i,j} \in \{0, 1\}$ with $\mathbf{Pr}[b_{i,j} = 1] = p_1^j$ for $X_i \in P_1$ and $\mathbf{Pr}[b_{i,j} = 1] = p_2^j$ for $X_i \in P_2$, such that $X_{i,j} = \frac{b+1}{2}$.

– Take top two left singular vectors u_1, u_2 of X, where $u_i = [u_{i,1}, \ldots, u_{i,n}], i = 1, 2$.

1. If $s_2(X) > T = \frac{15N}{32}\sqrt{3\omega_{\min}\gamma}$, use u_2 to partition the individuals with 0 as the threshold, i.e., partition $j \in [n]$ according to $u_{2,j} < 0$ or $u_{2,j} \geq 0$.
2. Otherwise, use u_1 to partition, with mixture mean $M = \sum_{i=1}^{n} u_{1,n}$ as the threshold.

Analysis of the Simple Algorithm: Our analysis is based on comparing entries in the top two singular vectors of the normalized random $n \times K$ matrix X, with those of a static matrix $\mathcal{X}$, where each entry $\mathcal{X}_{i,j} = \mathbf{E}[X_{i,j}]$ is the expected value of the corresponding entry in X. Hence $\forall i = 1, \ldots, N_1$, $\mathcal{X}_i = [\mu_1^1, \mu_1^2, \ldots, \mu_1^K]$, where $\mu_1^j = \frac{1+p_1^j}{2}, \forall j$, and $\forall i = N_1 + 1, \ldots, n$, $\mathcal{X}_i = [\mu_2^1, \mu_2^2, \ldots, \mu_2^K]$, where $\mu_2^j = \frac{1+p_2^j}{2}, \forall j$. We assume the divergence is exactly γ among the K features that we have chosen in all calculations.

The inspiration for this approach is based on the following lemma, whose proof is built upon a theorem that is presented in a lecture note by Spielman [22]. For a $n \times K$ matrix A, let $s_1(A) \geq s_2(A) \geq \ldots \geq s_n(A)$ be singular values of A. Let $u_1, \ldots, u_n, v_1, \ldots, v_n$, be the n left and right singular vectors of X, corresponding to $s_1(X), \ldots, s_n(X)$ such that $\|u_i\|_2 = 1, \|v_i\|_2 = 1, \forall i$. We denote the set of n left and right singular vectors of $\mathcal{X}$ with $\bar{u}_1, \ldots, \bar{u}_n, \bar{v}_1, \ldots, \bar{v}_n$.

Lemma 4. *Let X be the random $n \times K$ matrix and $\mathcal{X}$ its expected value matrix. Let $A = X - \mathcal{X}$ be the zero-mean random matrix. Let θ_i be the angle between two vectors: $[u_i, v_i]$, $[\bar{u}_i, \bar{v}_i]$, where $\|[u_i, v_i]\|_2 = \|[\bar{u}_i, \bar{v}_i]\|_2 = 2$ and $[u, v]$ represents a vector that is the concatenation of two vectors u, v.*

$$\|u_i - \bar{u}_i\|_2 \leq \|[u_i, v_i] - [\bar{u}_i, \bar{v}_i]\|_2 \approx 2\theta_i \approx 2\sin(\theta_i) \leq \frac{4s_1(A)}{\mathit{gap}(i, \mathcal{X})}, \tag{4}$$

where $\mathit{gap}(i, \mathcal{X}) = \min_{j \neq i} |s_i(\mathcal{X}) - s_j(\mathcal{X})|$.

We first bound the largest singular value $s_1(A) = s_1(X - \mathcal{X})$ of $(a_{i,j})$ with independent zero-mean entries, which defines the Euclidean operator norm

$$\|(a_{i,j})\| := \sup \left\{ \sum_{i,j} a_{i,j} x_i y_j : \sum x_i^2 \leq 1, \sum y_i^2 \leq 1 \right\}. \tag{5}$$

The behavior of the largest singular value of an $n \times m$ random matrices A with i.i.d. entries is well studied. Latala [17] shows that the weakest assumption for its regular behavior is boundedness of the fourth moment of the entries, even if they are not identically distributed. Combining Theorem 5 of Latala with the concentration Theorem 6 by Meckes [19] proves Theorem 7 that we need [1].

[1] One can also obtain an upper bound of $O(\sqrt{n+K})$ on $s_1(A)$ using a theorem on by Vu [24], through the construction a $(n+K) \times (n+K)$ square matrix out of A.

Theorem 5. (Bounded Norm of Random Matrices [17]) *For any finite $n \times m$ matrix A of independent mean zero r.v.'s $a_{i,j}$ we have, for an absolute constant C,*

$$\mathbf{E}\,\|(a_{i,j})\| \le C\left(\max_i \sqrt{\sum_j \mathbf{E}a_{i,j}^2} + \max_j \sqrt{\sum_i \mathbf{E}a_{i,j}^2} + \left(\sum_{i,j} \mathbf{E}a_{i,j}^4\right)^{\frac{1}{4}}\right). \quad (6)$$

Theorem 6. (Concentration of Largest Singular Value: Bounded Range [19]) *For any finite $n \times m$, where $n \le m$, matrix A, such that entries $a_{i,j}$ are independent r.v. supported in an interval of length at most D, then, for all t,*

$$\mathbf{Pr}[|s_1(A) - \mathbb{M}s_1(A)| \ge t] \le 4e^{-t^2/4D^2}. \quad (7)$$

Theorem 7. (Largest Singular Value of a Mean-zero Random Matrix) *For any finite $n \times K$, where $n \le K$, matrix A, such that entries $a_{i,j}$ are independent mean zero r.v. supported in an interval of length at most D, with fourth moment upper bounded by B, then*

$$\mathbf{Pr}\Big[s_1(A) \ge CB^{1/4}\sqrt{K} + 4D\sqrt{\pi} + t\Big] \le 4e^{-t^2/4} \quad (8)$$

for all t. Hence $\|A\| \le C_1 B^{1/4}\sqrt{K}$ for an absolute constant C_1.

2.1 Generating a Large Gap in $s_1(\mathcal{X}), s_2(\mathcal{X})$

In order to apply Lemma 4 to the top two singular vectors of X and $\mathcal{X}$ through

$$\|u_1 - \bar{u}_1\|_2 \le \frac{4s_1(X-\mathcal{X})}{|s_1(\mathcal{X}) - s_2(\mathcal{X})|} \quad (9)$$

$$\|u_2 - \bar{u}_2\|_2 \le \frac{4s_1(X-\mathcal{X})}{\min\left(|s_1(\mathcal{X}) - s_2(\mathcal{X})|, |s_2(\mathcal{X})|\right)}, \quad (10)$$

we need to first bound $|s_1(\mathcal{X}) - s_2(\mathcal{X})|$ away from zero, since otherwise, RHSs on both (9) and (10) become unbounded. We then analyze $\mathsf{gap}(2,\mathcal{X}) = \min\left(|s_1(\mathcal{X}) - s_2(\mathcal{X})|, |s_2(\mathcal{X})|\right)$.

Let us first define values a, b, c that we use throughout the rest of the paper:

$$a = \sum_{k=1}^{K} (\mu_1^k)^2, \quad b = \sum_{k=1}^{K} \mu_1^k \mu_2^k, \quad c = \sum_{k=1}^{K} (\mu_2^k)^2. \quad (11)$$

For the following analysis, we can assume that $a, b, c \in [K/4, K]$, given that X is normalized in Procedure Classify.

We first show that normalization of X as described in Procedure Classify guarantees that not only $|s_1(\mathcal{X}) - s_2(\mathcal{X})| \ne 0$, but there also exists a $\Theta(\sqrt{NK})$ amount of gap between $s_1(\mathcal{X})$ and $s_2(\mathcal{X})$ in Proposition 8:

$$\mathsf{gap}(\mathcal{X}) := |s_1(\mathcal{X}) - s_2(\mathcal{X})| = \Theta(\sqrt{NK}). \quad (12)$$

Proposition 8. *For a normalized random matrix X, its expected value matrix $\mathcal{X}$ satisfies $\frac{4c_0\sqrt{2NK}}{5} \leq \mathit{gap}(\mathcal{X}) \leq \sqrt{2NK}$, where $c_0 = \frac{|b|\sqrt{ac}}{K(a+c)}$ is a constant, given that $a, b, c \in [K/4, K]$ as defined in (11). In addition,*

$$\sqrt{\frac{KN}{4}} \leq s_1(\mathcal{X}) \leq \sqrt{2NK}, \text{ and } \sqrt{\frac{NK}{2}} \leq s_1(\mathcal{X}) + s_2(\mathcal{X}) \leq \sqrt{2NK}. \quad (13)$$

We next state a few important results that justify Procedure Classify. Note that the left singular vectors $\bar{u}_i, \forall i$ of $\mathcal{X}$ are of the form $[x_i, \ldots, x_i, y_i, \ldots, y_i]^T$:

$$\bar{u}_1 = [x_1, \ldots, x_1, y_1, \ldots, y_1]^T, \text{ and } \bar{u}_2 = [x_2, \ldots, x_2, y_2, \ldots, y_2]^T, \quad (14)$$

where x_i repeats N_1 times and y_i repeats N_2 times. We first show Proposition 9 regarding signs of $x_i, y_i, i = 1, 2$, followed by a lemma bounding the separation of x_2, y_2. We then state the key Separation Lemma that allows us to conclude that least one of top two left singular vectors of X can be used to classify data at each round. It can be extended to cases when $k > 2$.

Proposition 9. *Let b as defined in (11): when $b > 0$, entries x_1, y_1 in $\bar{u}_1$ have the same sign while x_2, y_2 in $\bar{u}_2$ have opposite signs.*

Lemma 10. $|x_2 - y_2|^2 \leq \frac{C_{\max}}{2N}$ *where* $C_{\max} = \left(\sqrt{\frac{1}{\omega_1}} + \sqrt{\frac{1}{\omega_2}}\right)^2 \leq \frac{4}{\omega_{\min}}$; $|x_2|^2 \geq \frac{C_{x\min}}{2N}$ *where* $C_{x\min} = \frac{\omega_2}{4\omega_1^2 + \omega_1\omega_2}$; $|y_2|^2 \geq \frac{C_{y\min}}{2N}$ *where* $C_{y\min} = \frac{\omega_1}{4\omega_2^2 + \omega_1\omega_2}$.

Lemma 11. (Separation Lemma) $K\gamma = s_1(\mathcal{X})^2(x_1 - y_1)^2 + s_2(\mathcal{X})^2(x_2 - y_2)^2$.

Proof. Let $\Delta := P_1 - P_2$ as in Theorem 2, and $\boldsymbol{b} = [1, 0, \ldots, 0, -1, 0, \ldots, 0]^T$, where 1 appears in the first and -1 appears in the $N_1 + 1^{st}$ positions. Then $\Delta = X^T\boldsymbol{b} = [\mu_1^1 - \mu_2^1, \mu_1^2 - \mu_2^2, \ldots, \mu_1^K - \mu_2^K]$. Given $\mathcal{X} = s_1(\mathcal{X})\bar{u}_1\bar{v}_1^T + s_2(\mathcal{X})\bar{u}_2\bar{v}_2^T$, we thus rewrite Δ as: $\Delta = \mathcal{X}^T\boldsymbol{b} = s_1(\mathcal{X})\bar{v}_1\bar{u}_1^T\boldsymbol{b} + s_2(\mathcal{X})\bar{v}_2\bar{u}_2^T\boldsymbol{b} = s_1(\mathcal{X})\bar{v}_1(x_1 - y_1) + s_2(\mathcal{X})\bar{v}_2(x_2 - y_2)$. The lemma follows from the fact that $\|\Delta\|_2 = \sqrt{K\gamma}$ and $\bar{v}_1, \bar{v}_2$ are orthonormal. ■

Combining Proposition 9, Lemma 10, (13), and Lemma 11, we have

Corollary 12. $s_2(\mathcal{X}) \leq \frac{\sqrt{2NK\gamma}}{\sqrt{c_{x\min}} + \sqrt{c_{y\min}}}$, *and hence* $\mathit{gap}(2, \mathcal{X}) = \min(s_2(\mathcal{X}), |s_1(\mathcal{X}) - s_2(\mathcal{X})|) = s_2(\mathcal{X})$ *for a sufficiently small* γ.

Finally, we show that the probability of error at each round for each individual is at most $f = 1/10$, given the sample size n as specified in Theorem 2. Hence by taking majority vote over the different runs for each sample, our algorithm will find the correct partition with probability $1 - 1/n^2$, given that at each round we take a set of $K > n$ independent features. We leave the detailed analysis in full version.

3 The Algorithm PARTITION

As in Section 2, by repeating the partitioning process $\log n$ times, we may restrict our attention to the problem of classifying a constant fraction of the individuals correctly.

Let $V = \{1, \ldots, n\}$ be the set of all n individuals, and let $\psi : V \to \{1, \ldots, k\}$ be the map that assigns to each individual the population it belongs to. Further, set $V_t = \psi^{-1}(t)$, define $N_t = |V_t|$, $\Gamma = K\gamma$, and $\lambda = \sqrt{K}\sigma$. In addition, let $A = (a_{vi})$ denote the empirical $n \times K$ input matrix. Then the assumption from Theorem 3 can be rephrased as $n_{\min} K\gamma > C_k\lambda^2$.

If $X = (x_{ij})_{1 \le i \le n, 1 \le j \le K}$ is a $n \times K$ matrix, then we let $\|X\| = \max_{\|\xi\|=1} \|X\xi\|$ signify the operator norm of X, while $\|X\|_F = (\sum_{i,j} x_{ij}^2)^{\frac{1}{2}}$ denotes the Frobenius norm. The algorithm PARTITION computes a *rank k approximation* $\widehat{A}$ of the input matrix A. That is, $\widehat{A}$ is a $n \times K$ matrix of rank at most k, and if B is any $n \times K$ matrix of rank at most k, then $\|A - \widehat{A}\| \le \|A - B\|$. Such an $\widehat{A}$ can be computed in polynomial time via singular value decomposition. Let $\widehat{A}_v$ denote the v-row of $\widehat{A}$.

Algorithm 13. PARTITION(A, k)
Input: A $n \times K$ matrix A and the parameter k. *Output:* A partition $S_1, \ldots, S_k$ of V.

1. Compute a rank k approximation $\widehat{A}$ of A.
   For $j = 1, \ldots, 2\log K$ do
2. Let $\Gamma_j = K2^{-j}$ and compute $Q^{(j)}(v) = \{w \in V : \|\widehat{A}_w - \widehat{A}_v\|^2 \le 0.01\Gamma_j^2\}$ for all $v \in V$.
   Then, determine sets $Q_1^{(j)}, \ldots, Q_k^{(j)}$ as follows: for $i = 1, \ldots, k$ do
3. Pick $v \in V \setminus \bigcup_{l=1}^{i-1} Q_l^{(j)}$ such that $|Q^{(j)}(v) \setminus \bigcup_{l=1}^{i-1} Q_l^{(j)}|$ is maximum.
   Set $Q_i^{(j)} = Q^{(j)}(v) \setminus \bigcup_{l=1}^{i-1} Q_l^{(j)}$ and $\xi_i^{(j)} = \frac{1}{|Q_i^{(j)}|} \sum_{w \in Q_i^{(j)}} \widehat{A}_w$.
4. Partition the entire set V as follows: first, let $S_i^{(j)} = Q_i^{(j)}$ for all $1 \le i \le k$.
   Then, add each $v \in V \setminus \bigcup_{l=1}^{k} Q_l^{(j)}$ to a set $S_i^{(j)}$ such that $\|\widehat{A}_v - \xi_i^{(j)}\|$ is minimum.
   Set $r_j = \sum_{i=1}^{k} \sum_{v \in S_i^{(j)}} \|\widehat{A}_v - \xi_i^{(j)}\|^2$.
5. Let J be such that $r^* = r_J$ is minimum. Return $S_1^{(J)}, \ldots, S_k^{(J)}$.

The basic idea behind PARTITION is to classify each individual $v \in V$ according to its row vector $\widehat{A}_v$ in the rank k approximation $\widehat{A}$. That is, two individuals v, w are deemed to belong to the same population iff $\|\widehat{A}_v - \widehat{A}_w\|^2 \le 0.01\Gamma^2$. Hence, PARTITION tries to determine sets $S_1, \ldots, S_k$ such that for any two v, w in the same set S_j the distance $\|\widehat{A}_v - \widehat{A}_w\|$ is small. To see why classifying the individuals according to their corresponding row vectors in $\widehat{A}$ is a good idea, we consider an auxiliary matrix $\mathbb{E} = (\mathbb{E}_{vi})$ with entries $\mathbb{E}_{vi} = p^i_{\psi(v)}$. Thus, the entries of $\mathbb{E}$ equal the expectations of the entries of A.

Lemma 14. *There is a constant $C > 0$ such that $\sum_{v \in V} \|\widehat{A}_v - \mathbb{E}_v\|^2 \le Ck\lambda^2$ whp.*

Proof. Recall that $\widehat{A}$ and $\mathbb{E}$ both have rank $\le k$, we obtain

$$\sum_{v \in V} \|\widehat{A}_v - \mathbb{E}_v\|^2 = \|\widehat{A} - \mathbb{E}\|_F^2 \le 2k\|\widehat{A} - \mathbb{E}\| \le 8k\|A - \mathbb{E}\|^2 \le Ck\lambda^2,$$

where the last inequality follows from Theorem 7. ∎

Observe that Lemma 14 implies that for *most* v we have $\|\widehat{A}_v - \mathbb{E}_v\|^2 \le 10^{-6}\Gamma$, say. For letting $z = |\{v : \|\widehat{A}_v - \mathbb{E}_v\|^2 > 10^{-6}\Gamma\}|$, we get $10^{-6}\Gamma z \le \sum_{v \in V} \|\widehat{A}_v - \mathbb{E}_v\|^2 \le$

$Ck\lambda^2$, whence $z \ll n_{\min}$ due to our assumption that $n_{\min}\Gamma \gg k\lambda^2$. Thus, most rows of $\widehat{A}$ are close to the corresponding rows of the *expected* matrix $\mathbb{E}$. Since Γ is not given to the algorithm as an input parameter, PARTITION has to estimate Γ on its own.

To this end, the outer loop goes through $2\log K$ "candidate values" Γ_j. These values are then used to obtain a partition $Q_1^{(1)}, \ldots, Q_1^{(k)}$ in Steps 2–4, which are similar to the algorithm presented in [18]. In addition, Step 4 computes the error parameter r_j. Finally Step 5 outputs the partition that minimizes the error parameter r_j. More precisely, Step 2 uses Γ_j to compute for each $v \in V$ the set $Q(v)$ of elements w such that $\|\widehat{A}_w - \widehat{A}_v\| \leq 0.01\Gamma_j^2$. Then, Step 3 tries to compute "big" disjoint $Q_1^{(j)}, \ldots, Q_k^{(j)}$, where each $Q_i^{(j)}$ results from some $Q(v_i)$. Further, Step 4 assigns all elements v not covered by $Q_1^{(j)}, \ldots, Q_k^{(j)}$ to that $Q_i^{(j)}$ whose "center vector" $\xi_i^{(j)}$ is closest to $\widehat{A}_v$.

Thus, we need to show that eventually picking the partition whose error term r_j is minimum yields a good approximation to the ideal partition $V_1, \ldots, V_k$. The basic reason why this is true is that $\xi_i^{(j)}$ should approximate the expectation $\mathbb{E}^{V_i}$ for class V_i well iff $Q_i^{(j)}$ is a good approximation of V_i. Hence, if $Q_1^{(j)}, \ldots, Q_k^{(j)}$ is "close" to $V_1, \ldots, V_k$, then $r_j = \sum_{i=1}^k \sum_{v \in S_i^{(j)}} \|\widehat{A}_v - \xi_i^{(j)}\|^2 \approx \|\widehat{A} - \mathbb{E}\|_F^2$ will be about as small as $\|\widehat{A} - \mathbb{E}\|_F^2$ (cf. Lemma 14). Furthermore, Lemma 16 shows that any partition such that r_j is small yields a good approximation of $V_1, \ldots, V_k$. Theorem 3 is an immediate consequence of Lemmas 15 and 16.

Lemma 15. *If $\frac{1}{2}\Gamma \leq \Gamma_j \leq \Gamma$, then $r_j \leq C_0 k^3 \lambda^2$ for a certain constant $C_0 > 0$.*

Lemma 16. *Let $S_1, \ldots, S_k$ be a partition and $\xi_1, \ldots, \xi_k$ a sequence of vectors such that $\sum_{i=1}^k \sum_{v \in S_i} \|\xi_i - A_v^*\|^2 \leq C_0 k^3 \lambda^2$. Then there is a bijection $\Xi : \{1, \ldots, k\} \to \{1, \ldots, k\}$ such that the following holds.*

1. *$\|\xi_i - \mathbb{E}^{V_{\Xi(i)}}\|^2 \leq 0.001\Gamma^2$ for all $i = 1, \ldots, k$, and*
2. *$\sum_{i=1}^k |S_i \triangle V_{\Xi(i)}| < 0.001 n_{\min}$.*

4 Experiments

We illustrate the effectiveness of spectral techniques using simulations. In particular, we explore the case when we have a mixture of two populations; we show that when $NK > 1/\gamma^2$ and $K > 1/\gamma$, either the first or the second left singular vector of X shows an approximately correct partitioning, meaning that the success rate is well above $1/2$. The entry-wise expected value matrix $\mathcal{X}$ is: among $K/2$ features, $p_1^i > p_2^i$ and for the other half, $p_1^i < p_2^i$ such that $\forall i, p_1^i, p_2^i \in \{\frac{1+\alpha}{2} + \frac{\epsilon}{2}, \frac{1-\alpha}{2} + \frac{\epsilon}{2}\}$, where $\epsilon = 0.1\alpha$. Hence $\gamma = \alpha^2$. We report results on balanced cases only, but we do observe that unbalanced cases show similar tradeoffs. For each population P, the success rate is defined as the number of individuals that are correctly classified, i.e., they belong to a group that P is the majority of that group, versus the size of the population $|P|$.

Each point on the SVD curve corresponds to an average rate over 100 trials. Since we are interested in exploring the tradeoffs of N, K in all ranges (e.g., when $N << K$

or $N >> K$), rather than using the threshold T in Procedure Classify that is chosen in case both $N, K > 1/\gamma$, to decide which singular vector to use, we try both u_1 and u_2 and use the more effective one to measure the success rate at each trial. For each data point, the distribution of X is fixed across all trials and we generate an independent $X_{2N\times K}$ for each trial to measure success rate based on the more effective classifier between u_1 and u_2.

One can see from the plot that when $K < 1/\gamma$, i.e., when $K = 200$ and 400, no matter how much we increase N, the success rate is consistently low. Note that $50/100$ of success rate is equivalent to a total failure. In contrast, when N is smaller than $1/\gamma$, as we increase K, we can always classify with a high success rate, where in general, $NK > 1/\gamma^2$ is indeed necessary to see a high success rate. In particular, the curves for $K = 5000, 2500, 1250$ show the sharpness of the threshold behavior for increasing sample size n from below $1/K\gamma^2$ to above. For each curve, we also compute the best possible classification one could hope to make if one knew in advance which features satisfied $p_1^i > p_2^i$ and which satisfied $p_1^i < p_2^i$. These are the horizontal(ish) dotted lines above each curve. The fact that the solid curves are approaching these information-theoretic upper bounds shows that the spectral technique is correctly using the available information.

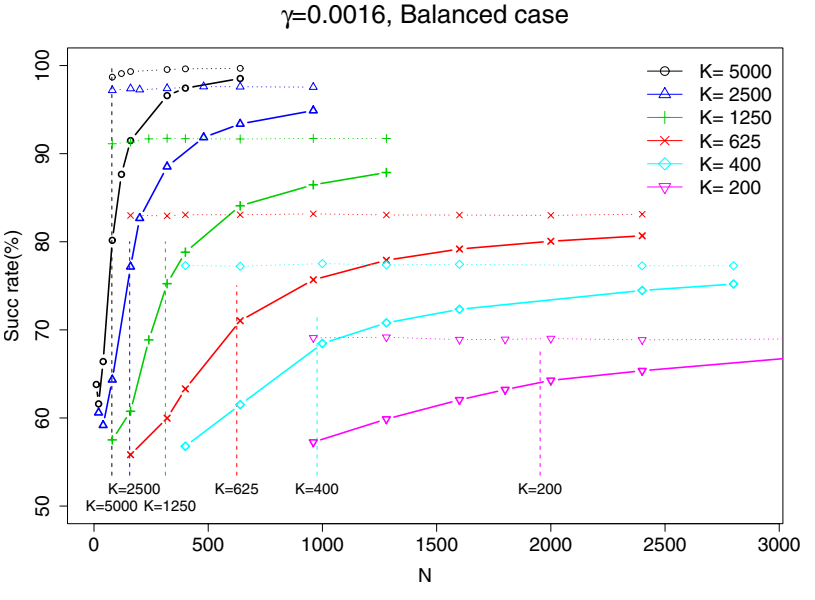


Fig. 1. Plots show success rate as a function of N for several values of K, when $\gamma = (0.04)^2$. Each point is an average over 100 trials. Horizontal lines ("oracles") indicate the information-theoretically best possible success rate for that value of K (how well one could do if one knew in advance which features satisfied $p_1^i > p_2^i$ and which satisfied $p_1^i < p_2^i$; they are not exactly horizontal because they are also an average over 100 runs). Vertical bars indicate the value of N for which $NK = 1/\gamma^2$.

Acknowledgments

We thank John Lafferty, Frank McSherry, Roman Vershynin and Larry Wasserman for many helpful discussions on this work.

References

1. Achlioptas, D., McSherry, F.: On spectral learning of mixtures of distributions. In: Auer, P., Meir, R. (eds.) COLT 2005. LNCS (LNAI), vol. 3559, pp. 458–469. Springer, Heidelberg (2005), http://www.cs.ucsc.edu/~optas/papers/
2. S. Arora and R. Kannan. Learning mixtures of arbitrary gaussians. In *Proceedings of 33rd ACM Symposium on Theory of Computing*, pages 247–257, 2001.
3. Chaudhuri, K., Halperin, E., Rao, S., Zhou, S.: A rigorous analysis of population stratification with limited data. In: Proceedings of the 18th ACM-SIAM SODA (2007)
4. Coja-Oghlan, A.: An adaptive spectral heuristic for partitioning random graphs. In: Bugliesi, M., Preneel, B., Sassone, V., Wegener, I. (eds.) ICALP 2006. LNCS, vol. 4051, Springer, Heidelberg (2006)
5. Cryan, M.: Learning and approximation Algorithms for Problems motivated by evolutionary trees. PhD thesis, University of Warwick (1999)
6. Cryan, M., Goldberg, L., Goldberg, P.: Evolutionary trees can be learned in polynomial time in the two state general markov model. SIAM J. of Computing 31(2), 375–397 (2002)
7. Dasgupta, A., Hopcroft, J., Kleinberg, J., Sandler, M.: On learning mixtures of heavy-tailed distributions. In: Proceedings of the 46th IEEE FOCS, pp. 491–500 (2005)
8. Dasgupta, S.: Learning mixtures of gaussians. In: Proceedings of the 40th IEEE Symposium on Foundations of Computer Science, pp. 634–644 (1999)
9. Dasgupta, S., Schulman, L.J.: A two-round variant of em for gaussian mixtures. In: Proceedings of the 16th Conference on Uncertainty in Artificial Intelligence (UAI) (2000)
10. Feldman, J., O'Donnell, R., Servedio, R.: Learning mixtures of product distributions over discrete domains. In: Proceedings of the 46th IEEE FOCS (2005)
11. Feldman, J., O'Donnell, R., Servedio, R.: PAC learning mixtures of Gaussians with no separation assumption. In: Lugosi, G., Simon, H.U. (eds.) COLT 2006. LNCS (LNAI), vol. 4005, Springer, Heidelberg (2006)
12. Fiedler, M.: Algebraic connectivity of graphs. Czechoslovak Mathematical Journal, 298–305 (1973)
13. Fjallstrom, P.: Algorithms for graph partitioning: a survey. Technical report, Linkoping University Electroni Press (1998)
14. Freund, Y., Mansour, Y.: Estimating a mixture of two product distributions. In: Proceedings of the 12th Annual COLT, pp. 183–192 (1999)
15. Kannan, R., Salmasian, H., Vempala, S.: The spectral method for general mixture models. In: Auer, P., Meir, R. (eds.) COLT 2005. LNCS (LNAI), vol. 3559, Springer, Heidelberg (2005)
16. Kearns, M., Mansour, Y., Ron, D., Rubinfeld, R., Schapir, R., Sellie, L.: On the learnability of discrete distributions. In: Proceedings of the 26th ACM STOC, pp. 273–282 (1994)
17. Latala, R.: Some estimates of norms of random matrices. In: Proceedings of the American Mathematical Society, vol. 133, pp. 1273–1282 (2005)
18. McSherry, F.: Spectral partitioning of random graphs. In: Proceedings of the 42nd IEEE Symposium on Foundations of Computer Science, pp. 529–537 (2001)
19. Meckes, M.: Concentration of norms and eigenvalues of random matrices. J. Funct. Anal. 211(2), 508–524 (2004)

20. Mossel, E., Roch, S.: Learning nonsinglar phylogenies and hidden markov models. In: Proceedings of the 37th ACM STOC (2005)
21. Pritchard, J.K., Stephens, M., Donnelly, P.: Inference of population structure using multilocus genotype data. Genetics 155, 954–959 (2000)
22. Spielman, D.: The behavior of algorithms in practice, Lecture notes (2002)
23. Vempala, V., Wang, G.: A spectral algorithm of learning mixtures of distributions. In: Proceedings of the 43rd IEEE FOCS, pp. 113–123 (2002)
24. Vu, V.: Spectral norm of random matrices. In: Proceedings of 37th ACM STOC, pp. 423–430 (2005)
25. Zhou, S.: Routing, Disjoint Paths, and Classification. PhD thesis, Carnegie Mellon University, Pittsburgh, PA, CMU Technical Report, CMU-PDL-06-109 (2006)

A Constant-Competitive Algorithm for Online OVSF Code Assignment

F.Y.L. Chin*, H.F. Ting**, and Y. Zhang***

Department of Computer Science
The University of Hong Kong
Pokfulam Road, Hong Kong

Abstract. Orthogonal Variable Spreading Factor (OVSF) code assignment is a fundamental problem in Wideband Code-Division Multiple-Access (W-CDMA) systems, which plays an important role in third generation mobile communications. In the OVSF problem, codes must be assigned to incoming call requests with different data rate requirements, in such a way that they are mutually orthogonal with respect to an OVSF code tree. An OVSF code tree is a complete binary tree in which each node represents a code associated with the combined bandwidths of its two children. To be mutually orthogonal, each leaf-to-root path must contain at most one assigned code. In this paper, we focus on the online version of the OVSF code assignment problem and give a 10-competitive algorithm, improving the previous $O(h)$-competitive result, where h is the height of the code tree, and also another recent constant-competitive result, where the competitive ratio is only constant under amortized analysis and the constant is never determined. Finally, we also improve the lower bound of the problem from 3/2 to 5/3.

1 Introduction

Wideband Code-Division Multiple-Access (W-CDMA) technology is one of the main technologies widely-developed in recent years for the implementation of third-generation (3G) cellular systems. We consider the well-studied problem of Orthogonal Variable Spreading Factor (OVSF) code assignment in W-CDMA systems [5, 7, 11, 12, 14].

OVSF is an implementation of CDMA wherein, before each signal is transmitted, the spectrum is spread according to a unique code, which is derived from an OVSF code tree. An OVSF code tree is a complete binary tree. Users have requests for different data rates, and the OVSF code tree accommodates these different requests by assigning codes at different levels of the code tree, with the root being at the highest level and representing the entire bandwidth of the

* Supported by HK RGC grant HKU-7113/07E.
** Supported by HK RGC grant HKU-7045/02E.
*** Supported by the European Regional Development Fund (ERDF). Current address: Institut für Mathematik, Technische Universität, Berlin.

T. Tokuyama (Ed.): ISAAC 2007, LNCS 4835, pp. 452–463, 2007.

wireless system. The code at any node other than the root denotes the bandwidth half that of its parent in the tree. In any *legal assignment* in the code tree, no two assigned codes lie on a single path from the root to a leaf, i.e., any two assigned codes are *mutually orthogonal*. The subset of nodes in the code tree, which forms a legal assignment, is called a *code assignment*. A node x is said to be *free* if there are no assigned nodes in every root-to-leaf path containing x, and thus making x an assigned node would still result in a legal assignment. For convenience, we use the words "code" and "node" interchangeably. Fig. 1 is an example of an OVSF code tree with the code assignment represented by the darkened nodes marked as c, d, e, g and i.

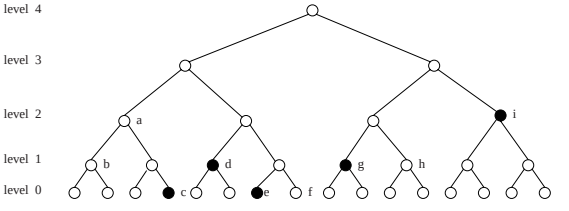


Fig. 1. An example of OVSF code tree, solid circles are the assigned codes

To illustrate the essence of the OVSF code assignment problem, consider the configuration shown in Fig. 1. Let $Req(x)$ denote the request to which code x is assigned. Suppose a level-1 code request arrives followed by a level-2 code request. If code b were assigned to the first request, we would have to make two code reassignments before we can assign code a to the second request, e.g. assign code h to $Req(b)$ (and thereby freeing b) and assign f to $Req(c)$ (freeing c and consequently a). If, on the other hand, h were assigned to the first request, only one reassignment would be needed to satisfy the second request, i.e. assign f to $Req(c)$, then assign code a to the second request.

Since each reassignment requires processing overhead and may affect the quality of communications, a natural idea is to design algorithms to minimize the number of reassignments. Note that this problem will not be too difficult and can be solved optimally by a greedy strategy if codes were never released. However, when codes can be released, the code tree can be fragmented and many reassignments might be needed if a good assignment algorithm was not used.

In general, an algorithm for OVSF code assignment is expected to handle a sequence $\sigma = (C_1, C_2, \ldots, C_k, \ldots)$ of code operations over time, each operation C_k being either to *request* a code at a particular level or to *release* an assigned code. Note that, if the total bandwidth of any set of free codes is less than the bandwidth required by a code request, the new code request has to be withdrawn.

The OVSF code assignment problem is hard, and the approach has often been to produce heuristics, whose performance is measured by the approximation (or competitive) ratio, which compares the cost of the algorithm to the cost of the optimal off-line scheme, where cost is the total number of assignments or

reassignments done by the algorithm. The problem has been studied extensively recently. There are several variations of this problem, including:

One-Step Off-line Code Assignment: Given a code assignment F and a new level-i code request r, find a code assignment F', which satisfies the new request with a minimal number of reassignments. For this variation, Minn and Siu [11] gave a greedy algorithm, and Erlebach *et al.* [5] proved that this problem is NP-hard and gave an $O(h)$-approximation algorithm, where h is the height of the OVSF code tree.

General Off-line Code Assignment: Given a sequence σ of code operations, find a sequence of code assignments such that the total number of reassignments is minimum, assuming the initial code tree is empty. This variation was proved to be NP-hard by Tomamichel [13], who also gave an exponential-time algorithm for this variation.

Online Code Assignment: The operations $C_1, C_2, C_3, \ldots$ in the sequence $\sigma = (C_1, C_2, \ldots, C_k, \ldots)$ arrive through time. At any time $t > 0$, we only know about the operations until t and have no information about any future operations $C_{t'}$ with $t' > t$. Again, the problem is to find a sequence of code assignments such that the total number of reassignments is minimum. For this variation, Erlebach *et al.* [5] gave an $O(h)$-competitive algorithm, where h is the height of the code tree. With resource augmentation, which means the online algorithm is allowed to use more bandwidth than the optimal scheme, a 4-competitive algorithm with a double-sized code tree was given in [5]. Using 1/8 extra bandwidth (less resource augmentation), Chin *et al.* [3] gave an 5-competitive algorithm. Recently, Forišek *et al.* [6] gave an online algorithm whose competitive ratio is constant under amortized analysis. In their paper, there is no estimate about the size of the constant and the worst case can still be $O(h)$.

In this paper, we focus on the online OVSF code assignment problem. We first observe that the online algorithm in [5], which is $O(h)$-competitive, forces the assigned codes in the OVSF code tree into a single fixed format. As observed in [5], there are two worst-case single-format-respecting configurations which make the performance of the algorithm in [5] poor, one which is bad (i.e. requires a reassignment at each level of the OVSF code tree) for code request but good (i.e. constant reassignments) for code release and the other which is bad for code release but good for code request. Interestingly, these two code configurations differ by only one code assignment (but differ much in their structure), and so there exists a sequence of alternating code requests and releases, each of which requires h code reassignments, and hence $O(h)$ for the competitive ratio. By allowing two types of format with similar structure, we are able to give a 10-competitive algorithm, improving the previous $O(h)$-competitive result [5], and the amortized $O(1)$-competitive result [6].

The rest of this paper is organized as follows. Sections 2 and 3 give the preliminary and the basic idea of our algorithm. Section 4 introduces a 10-competitive algorithm. Section 5 gives the correctness proof of the algorithm. A new lower bound of the problem, improving the bound from 3/2 to 5/3, is presented in Section 6.

2 Preliminaries

Let T be an OVSF code tree with a legal assignment A. In our discussion, we assume that T is ordered. We say that node u is *dead* if either it is assigned or at least one of its descendent is assigned. We say that a level ℓ of T is *compact* if any node at level ℓ that is to the left of some dead node at ℓ is also dead. We say that A is compact if all levels of T are compact. The following lemma suggests that the assigned nodes in a compact assignment are sorted; if we scan the OVSF code tree from left to right, the levels of the assigned nodes are non-decreasing.

Lemma 1. *Suppose that the legal assignment A is compact. Let u be an assigned node at level i and v be an assigned node at level j where $i < j$. Then, the level-j ancestor a_u of u must be to the left of v.*

Proof. Since A is legal, a_u cannot be v. If a_u is to the right of v, the dead node u has some nodes to its left, namely the level-i descendents of v, that are not dead; a contradiction.

Intuitively, we should make the assignments compact in order to fully utilize the bandwidth. There is a simple strategy to ensure compactness: To serve a level-ℓ code request r, we "append" it to the right-end of the list of dead nodes at ℓ, or more precisely, we assign to r the node u that is immediately after the last dead node, i.e., the rightmost dead node at ℓ. It is obvious that the resulting assignment is also compact. However, it may not be legal; although u does not have any assigned descendent (because it is not dead before the update), it may have some assigned ancestor. To solve the problem, we distinguish two kinds of levels. Consider any level ℓ. Let u be the node immediately after the last dead node at ℓ; if ℓ does not have any dead node, then let u be the leftmost node at ℓ. We say that ℓ is *rich* if u is free, i.e., the node does not have any ancestor or descendent that is assigned; otherwise, ℓ is *poor*. (Note that a level may be poor even if no nodes at ℓ are assigned.) It is easy to verify that if ℓ is rich, then the resulting assignment is still legal after assigning u to r. Suppose that ℓ is poor. Then, u is not free and it must have an ancestor v assigned to some request $Req(v)$. After assigning u to r, we need to reassign $Req(v)$, i.e., freeing v followed by a code request $Req(v)$, to make sure the assignment is legal. Note that this may trigger other reassignments of requests at higher levels.

The algorithm below describes how this simple approach serves a level-ℓ request r. It makes use of two procedures AppendRich and AppendPoor. Let u be the node immediately after the last dead node at level ℓ (if there is no dead node at ℓ, then u is its leftmost node). Procedure AppendRich(ℓ, r) is used when ℓ is rich; it simply assigns u to request r. Procedure AppendPoor(ℓ, r) is for the case when ℓ is poor. After assigning u to r, AppendPoor(ℓ, r) frees the assigned ancestor a of u, and returns the request to which a is assigned before it is freed.

1: **while** ℓ is poor **do**
2: $\quad r_g =$ AppendPoor(ℓ, r); $\{r_g$ is a level-g request.$\}$
3: $\quad \ell = g$; $r = r_g$;

4: **end while**
5: AppendRich(ℓ, r);

3 Some Ideas for Improvement

Note that the simple algorithm given in Section 2 might make a large number of calls to AppendPoor, and hence make a lot of (re)assignments, to serve a request. The following lemma is the key for reducing the number of (re)assignments.

Lemma 2. *Suppose ℓ is poor. After executing* AppendPoor(ℓ, r)*, ℓ becomes rich.*

Proof. Note that if the last dead node u at ℓ is the left son of its parent p, then ℓ must be rich. It is because the right son v of p, which is immediately after u, must be free; v is not dead (because u is the last dead node) and hence has no assigned descendent, and u and v share the same set of ancestors and thus v does not have any assigned ancestor. Thus, if ℓ is poor, node u must be a right son. After AppendPoor(ℓ, r), the node after u, which is a left son, becomes the new last dead node of ℓ. As argued above, ℓ becomes rich.

Note that if we call AppendPoor m times, then m levels will be changed from poor to rich. This is good because the next time we serve any request on these rich levels, we just need a simple assignment. The problem is that there might be no more requests on these levels, and the effort is wasted. To overcome the difficulty, we propose a lazy approach. Here is the idea. Suppose that there is a level-ℓ request r, and the levels $\ell, \ell+1, \ldots, k-1$ are all poor, and level k is rich. Then, we will call AppendPoor $k-\ell$ times and then call AppendRich once. Our lazy approach will not make these $k-\ell$ calls for AppendPoor; instead, it jumps to the last step, calling AppendRich to assign the node u after the last dead node at k to the level-ℓ request r.[1] Later, if there is some request on some level $g \in [\ell, k]$, we will do the necessary work that we have avoided previously in order to recover the correct free node at g and assign it to the request. To summarize, the lazy approach also aims at maintaining compact assignments. However, there may be some nodes that are assigned to some lower-level requests; we call these nodes *partially assigned nodes*. These partially assigned nodes induce some structures called *tanks of free nodes*, or simply *tanks*, which are intervals $[\ell, k]$ of levels with the following properties: The level $\ell, \ell+1, \ldots, k-1$ are all poor and the assigned nodes at these levels are all fully assigned. For level k, the last dead node of k is partially assigned to a level-ℓ request, and the remaining assigned nodes are fully assigned. We say that ℓ is the bottom of the tank $[\ell, k]$, and k is its top.

It is not difficult to implement the lazy approach in such a way that the number of (re)assignments needed for serving a request can be reduced substantially. However, to achieve a constant number of (re)assignments, we need to impose two extra structural properties on tanks. The first property is about the top of

[1] We are generous here by assigning a level-k node to a level-ℓ request. If we insist that a level-ℓ node must be assigned to r, then we can actually assign a level-ℓ descendent of u to r.

tanks. Given any level ℓ, we say that ℓ is *locally rich* if the last dead node at ℓ is a left son of its parent. The following fact is easy to verify from the definition and the proof of Lemma 2.

Fact 1. *A locally rich level is a rich level. Suppose ℓ is locally rich. Then after executing* AppendRich(ℓ, r)*, ℓ is no longer locally rich. If ℓ is poor, then after executing* AppendPoor(ℓ, r)*, ℓ becomes locally rich.*

The first property on tanks that we need to maintain is the following:

(†) The top of every tank must be locally rich.

To describe the second structural property, we consider two tanks $[b_o, t_o]$ and $[b_1, t_1]$. Suppose that $[b_o, t_o]$ is below $[b_1, t_1]$, i.e, $t_o < b_1$. We say that the two tanks are *merge-able* if (i) all the levels between them, i.e., levels $t_o + 1, t_o + 2, \ldots, b_1 - 1$, are empty, i.e., the levels do not have any assigned nodes, and (ii) the last dead node at level t_o is the leftmost level-t_o descendent of its level-b_1 ancestor. We find that merge-able tanks are bad for our approach. Therefore, our updating procedure will merge any two merge-able tanks as soon as they appear. In other words, our algorithm keeps the following invariant:

(∗) There is no merge-able tank in the assignment.

As can be seen in Figure 2, it is easy to merge two merge-able tanks using two (re)assignments, and the merging preserves the compactness of the assignment.

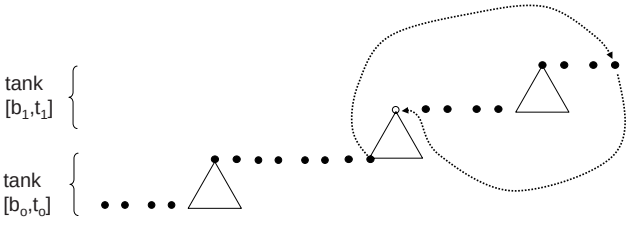


Fig. 2. Merging of two tanks

4 A Lazy Algorithm

In this section, we describe the algorithm LAZY, which implements the lazy approach efficiently. To simplify the description, we regard a locally rich level ℓ that does not belong to any tank is a tank $[\ell, \ell]$ itself. Furthermore, when there is no confusion, we will add a subscript to a request to indicate its level, e.g., the request r_g is a level-g request. In addition to AppendPoor and AppendRich, LAZY also makes use of the following two procedures.

– `FreeTail`(ℓ): The level ℓ must not be empty. The procedure frees the last assigned node u at ℓ and returns the request to which u is assigned.

– `ReAssign`(r, r'): Here, r is a request in the assignment, and r' is one that is not in the assignment. Suppose that u is assigned to r. Then, the procedure frees u from r, and then assigns u to r'.

We are now ready to describe LAZY. We first describe how it serves a code request. Then, we explain how it releases a code.

4.1 Serving a Level-ℓ Request r_ℓ

We have three different cases to consider.

Case 1: ℓ is poor and does not belong to any tank. Let h be the lowest level above ℓ that either belongs to some tank, or is rich. If there is no such h, we report not enough bandwidth (and we will prove in Section 5 that this is true). Note that if h does not belong to any tank, then h is rich, but not locally rich (otherwise, $[h, h]$ itself is a tank). In such case, we simply call AppendRich(h, r_ℓ). It can be verified that afterwards, h is locally rich and $[\ell, h]$ becomes a tank. To ensure $(*)$, we merge, if there is any, the merge-able tanks above or below $[\ell, h]$. Thus, the total number of (re)assignments is at most 5. The case when h belongs to some tank is more complicated. In such case, h must be the bottom of this tank $[h, t]$. By definition, the last assigned node at level t is partially assigned to a level-h request r_h. To serve r_ℓ, we first recover the free node at h by re-assigning r_h back to h. Then, we insert r_ℓ to a level lower than h so that the (re)assignments can make use of the free node at h and stop.

1: **if** $h \neq t$ **then**
2: $\quad r_h =$ `FreeTail`(t); $\{t$ becomes rich$\}$
3: $\quad r_g =$ AppendPoor(h, r_h); AppendRich(t, r_g); $\{[g, t]$ becomes tank$\}$
4: **end if**
5: {At this point, h is not empty and is locally rich (Fact 1)}
6: Let k be the highest level below h that is not empty.
7: **If** $(k < \ell)$ **then** let $k = \ell$.
8: $s =$ AppendPoor(k, r_ℓ); $\{s$ must be from h and $[\ell, k]$ becomes tank.$\}$
9: AppendRich(h, s);
10: {From Fact 1, h is not locally rich and thus $[h, h]$ is not a tank.}

It can be verified that the update needs at most 4 (re)assignments. Note that the additional tanks $[g, t]$ and $[\ell, k]$ may be created. To ensure $(*)$, we need to do some tank mergings. As pointed out in Line 10, $[h, h]$ is not a tank. Furthermore, there is no merge-able tank above $[g, t]$ (because $(*)$ ensures there is none above $[h, t]$ before the update). Therefore, we only need to merge tank below $[\ell, k]$, which requires two extra (re)assignments. Thus, we need at most 6 assignments to serve request r. However, if we need to merge $[\ell, k]$ with a tank below, we can save the assignment used by AppendPoor(k, r_ℓ) at Line 8; r_ℓ will be reassigned during the merging. This reduces the number of (re)assignments to 5.

Case 2: ℓ is poor and belongs to some tank $[b, t]$. For this case, we insert r_ℓ to ℓ directly, and the tank $[b, t]$ may be broken into two tanks, one above, and one below ℓ.

1: $r_g = \text{AppendPoor}(\ell, r_\ell)$; {$\ell$ becomes locally rich.}
2: $r_b = \texttt{FreeTail}(t)$; $\text{AppendRich}(t, r_g)$; {$[g, t]$ becomes tank.}
3: {We have served r_ℓ successfully, but there is no node assigned to r_b}
4: **if** $b = \ell$ **then**
5: $\quad\text{AppendRich}(\ell, r_b)$; {Now, ℓ is not locally rich}
6: **else**
7: $\quad$Let k be the highest level below ℓ that is not empty.
8: $\quad$**If** $(k < b)$ **then** let $k = b$.
9: $\quad s = \text{AppendPoor}(k, r_b)$; {$s$ must be from ℓ and $[b, k]$ becomes tank.}
10: $\quad\text{AppendRich}(\ell, s)$; {Now, ℓ is not locally rich.}
11: **end if**

It can be verified that the total number of (re)assignments made is at most 4. Since ℓ is not locally rich, $[\ell, \ell]$ is not a tank, and as guaranteed by $(*)$, there is no merge-able tanks above t or below b before the update. Therefore, $[g, t]$ and $[b, k]$ have no merge-able tank above or below them, and this case does not need any tank merging.

Case 3: ℓ is rich. For this case, we do the followings.

1: **if** ℓ does not belong to any tank **then**
2: $\quad\text{AppendRich}(\ell, r_\ell)$;
3: **else**
4: $\quad${for this case ℓ belongs to a tank, and since ℓ is rich, it is the top of some tank $[b, \ell]$.}
5: $\quad$**if** $b = \ell$ **then**
6: $\quad\quad\text{AppendRich}(\ell, r_\ell)$
7: $\quad$**else**
8: $\quad\quad r_b = \texttt{FreeTail}(\ell)$; $\text{AppendRich}(\ell, r_\ell)$;
9: $\quad\quad$Let k be the highest level below ℓ that is not empty.
10: $\quad\quad$**If** $(k < b)$ **then** let $k = b$.
11: $\quad\quad s = \text{AppendPoor}(k, r_b)$; $\text{AppendRich}(\ell, s)$;
12: $\quad$**end if**
13: **end if**

It can be verified that after the possible merging of tanks, the total number of (re)assignments made is at most 5.

Note that our algorithm uses AppendPoor and AppendRich to append a request after the last dead node of a level, and whenever we free a node, we will immediately assign it to some other request. Therefore, the compactness of the resulting assignment is preserved.

4.2 Release of a Level-ℓ Node Assigned to Request r

We first consider the case when ℓ is not in any tank. Then, ℓ is not locally rich; otherwise, $[\ell, \ell]$ itself is a tank. For this case, we do the following:

1: $r_\ell = \texttt{FreeTail}(\ell)$;
2: **if** $r \neq r_\ell$ **then** $\texttt{ReAssign}(r, r_\ell)$;

From the fact that ℓ is not locally rich before the update, it can be verified that the resulting assignment is still compact. Together with the two possible tank mergings, the update makes at most 5 (re)assignments. We now consider the case when ℓ is in some tank $[b, t]$. To release r, we do the following:

1: $r_b = \texttt{FreeTail}(t)$;
2: **if** $b = \ell$ **then**
3: **if** $r \neq r_b$ **then** $\texttt{ReAssign}(r, r_b)$;
4: **else**
5: Let k be the highest level below ℓ that is not empty.
6: **If** $(k < b)$ **then** let $k = b$.
7: $s = \text{AppendPoor}(k, r_b)$; {$s$ must be from ℓ, and $[b, k]$ becomes tank.}
8: **if** $r \neq s$ **then** $\texttt{ReAssign}(r, s)$;
9: **end if**

Note that after the execution, the assignment may not be compact; we have freed the last assigned node at level t and did not reassign the node to any request. This may create some hole, i.e., free node, between two dead nodes at some levels above t. In such case, we need to fill up the hole as follows. Let z be the lowest level above t that is not empty, and $v_{\texttt{hole}}$ be the free node, if any, created at level z. Note that by $(*)$, there is no merge-able tank above $[b, t]$; this implies z is not in any tank, and it is not locally rich. The following two steps will restore the compactness of the assignment:

1: $r_z = \texttt{FreeTail}(z)$; {$z$ is now locally rich}
2: Assign $v_{\texttt{hole}}$ to r_z.

It is important to note that freeing the last assigned node u at level z will not create any problem; since z is not locally rich, u must be the right son of its parent p. After freeing u, p is still dead because its left son is dead. It can be verified that the whole release procedure, together with possible tank mergings, uses at most 5 (re)assignments in the worst case.

The following theorem summarizes our discussion in this section.

Theorem 1. *Let A be a compact assignment satisfying $(*)$. LAZY serves any code request or code release for A using at most 5 (re)assignments. The resulting assignment is still compact and satisfies $(*)$. Furthermore, LAZY is 10-competitive.*

Proof. To see that LAZY is 10-competitive, suppose that there are m_1 code requests and m_2 code releases. Obviously, $m_2 \leq m_1$. To serve these requests and releases, LAZY makes at most $5m_1 + 5m_2 \leq 10m_1$ (re)assignments. Note that the optimal algorithm has to make at least m_1 assignments for the m_1 code requests. The theorem follows.

5 LAZY Fully Utilizes the Bandwidth

In this section, we prove that LAZY fully utilizes the bandwidth. More precisely, we prove that if LAZY cannot find an assignment to satisfy all the requests, then

no assignment can satisfy these requests. First, we need to define the notion of *leaf capturing.* A node that is fully assigned captures all of its leaf descendents. A node that is partially assigned to a level-ℓ request captures its leftmost 2^ℓ leaf descendents. For any node u, define $F(u)$ to be the set of leaf descendents of u that are not captured. For any set X of nodes, define $F(X) = \sum_{u \in X} F(u)$. Intuitively, for the root r, $F(r)$ is the remaining bandwidth not used by the current assignment. It is important to note that for a fixed set L of code requests, different legal assignments for L have the same value of $F(r)$.

Lemma 3. *Let A be an assignment maintained by LAZY, and for any level ℓ, let D_ℓ be the set of dead nodes at level ℓ. Then, $|F(D_\ell)| < 2^\ell$.*

Proof. The lemma is obviously true for level 0. Suppose it is true for all levels below ℓ, and we consider the level ℓ. Let $u_1, u_2, \ldots, u_k$ be the sequence of dead nodes at ℓ where u_k is the last dead node. First, we assume that there is no partially assigned node at ℓ. Let u_i be the last node that is not assigned. Then, $F(D_\ell) = F(\{u_1, u_2, \ldots u_i\}) + F(\{u_{i+1}, \ldots, u_k\})$. Since u_i is the last node not assigned, $u_{i+1}, \ldots, u_k$ are all assigned and the second term $F(\{u_{i+1}, \ldots, u_k\})$ is zero. To estimate the first term, let w be the last dead node at level $\ell - 1$. Note that w must be a child of u_i; it cannot be a child of the nodes to the right of u_i because these nodes are either assigned or not dead, and it cannot be a child of $u_1, \ldots, u_{i-1}$, otherwise u_i is not dead (recall that u_i is not assigned). If w is the right child of u_i, then $F(\{u_1, \ldots, u_i\}) = F(D_{\ell-1})$; otherwise $F(\{u_1, \ldots, u_i\}) = F(D_{\ell-1}) + 2^{\ell-1}$. Together with the induction hypothesis that $F(D_{\ell-1}) < 2^{\ell-1}$, the lemma follows.

We now suppose that there is a partially assigned node at ℓ. According to LAZY, we conclude that the last dead node u_k is the only partially assigned node at ℓ. Suppose that it is partially assigned by a level-g request. Then, $[g, \ell]$ is a tank and the levels $g, g+1, \ldots, \ell-1$ are all poor. Let $w_1, w_2, \ldots, w_m$ be the sequence of level-g descendents of $u_1, u_2, \ldots, u_{k-1}$. Then, $F(D_\ell) = F(u_1, \ldots, u_{k-1}) + F(u_k) = F(w_1, \ldots, w_m) + F(u_k) = F(w_1, \ldots, w_i) + F(w_{i+1}, \ldots, w_m) + F(u_k)$ where w_i is the last dead node at level g. By the induction hypothesis, we conclude that $F(D_g) = F(\{w_1, \ldots, w_i\}) < 2^g$, and by the definition of captured leaves for partially assigned node, we have $F(u_k) = 2^\ell - 2^g$. In the rest of the proof, we show that $F(w_{i+1}, \ldots, w_m) = 0$ and the lemma follows.

Suppose to the contrary that $F(w_{i+1}, \ldots, w_m) > 0$. Then, among the leaf descendents of $w_{i+1}, \ldots, w_m$, there is one that is not captured. Let w and u be respectively the level-g and level-ℓ ancestors of this leaf. Note that w is to the right of w_i and hence is not dead, and u is to the left of u_k and hence is dead . Let d be the last dead node along the path from u down to w. Suppose that d is at level h. It follows that its left child $v_{\texttt{left}}$ must be dead, and its right child $v_{\texttt{right}}$, which must be along the path from u to w, is not dead (because d is the last dead node on this path). Since $v_{\texttt{left}}$ is dead, none of its ancestors is assigned, and since $v_{\texttt{right}}$ is not dead, none of its descendent is assigned. Note that $v_{\texttt{left}}$ and $v_{\texttt{right}}$ has the same set of ancestors and it follows that $v_{\texttt{right}}$ is free. Therefore, we conclude that at level $h-1$, where $g \leq h-1 \leq \ell-1$, there

is a free node $v_{\texttt{right}}$ following the dead node $v_{\texttt{left}}$, and thus $h-1$ is not poor; a contradiction.

Theorem 2. *Suppose that LAZY reports "not enough bandwidth" when serving a level-ℓ request r. Let L be the set of requests in the current assignment. Then, there is no assignment that can safisfy all the requests in $L \cup \{r\}$.*

Proof. LAZY reports "not enough bandwidth" because level ℓ, as well as all levels above ℓ are poor. Let $u_1, u_2, \ldots, u_i$ be the sequence of dead nodes, and $u_{i+1}, \ldots, u_m$ be the remaining nodes, at ℓ. Then there are $F(\{u_1, \ldots, u_i\}) + F(\{u_{i+1}, \ldots, u_m\})$ leaves that are not captured. By Lemma 3, we conclude that $F(\{u_1, \ldots, u_i\}) < 2^\ell$. Since all levels above ℓ are poor, we can use an argument similar to the one used in the proof of Lemma 3 that $F(\{u_{i+1}, \ldots, u_m\}) = 0$. It follows that $F(\{u_1, u_2 \ldots, u_m\} < 2^\ell$. Since assigning any node to r needs to capture 2^ℓ leaves, there is no assignment that can satisfy all requests $L \cup \{r\}$.

6 Lower Bound

In [5], it is shown that the competitive ratio of any online algorithm for the code assignment problem must be at least 1.5. The following theorem shows that this bound can be improved to 5/3 by modifying the main idea given in [5].

Theorem 3. *No deterministic algorithm can solve the online code assignment problem better than 5/3-competitive.*

Proof. Consider a code tree with N leaves (leaf-codes) and a sequence of level-0 code requests with each request assigned to each leaf code one by one. As soon as both right and left subtrees of the root have at least $N/4$ assigned leaf codes, the adversary will stop issuing any more level-0 code requests. Thus there will be at most $3N/4$ level-0 code requests in the sequence. Then the adversary will repeatedly release those requests in the subtree with more than $N/4$ assigned leaf codes until both subtrees have exactly $N/4$ assigned leaf codes. The adversary will then make a level-$(n-1)$ request which will cause at least $N/4$ code reassignments, which end up with either the right or the left subtree with full assigned leaf codes. The adversary will then proceed recursively with the subtree with full assigned leaf codes by releasing its every other node. This process will be repeated $\log_2 N - 1$ times with a total of $N/2 - 1$ reassignments. On the other hand, the optimal algorithm can assign the leaf codes in such a way that no extra reassignments will be needed. Thus the optimal algorithm will take no more than $3N/4 + \log_2 N - 1$ assignments, whereas the adversary will take a total of $5N/4 + \log_2 N - 2$ (re)assignments. As a consequence, the competitive ratio will tend to be 5/3 asymptotically.

Acknowledgements. The authors thank Dr. Bethany M. Y. Chan for her efforts in making this paper more readable.

References

1. Chan, W.T., Chin, F.Y.L., Ye, D., Zhang, Y., Zhu, H.: Frequency Allocation Problem for Linear Cellular Networks. In: Asano, T. (ed.) ISAAC 2006. LNCS, vol. 4288, pp. 61–70. Springer, Heidelberg (2006)
2. Chan, W.T., Chin, F.Y.L., Ye, D., Zhang, Y., Zhu, H.: Greedy Online Frequency Allocation in Cellular Networks. Information Processing Letters 102(2-3), 55–61 (2007)
3. Chin, F.Y.L., Zhang, Y., Zhu, H.: Online OVSF Code Assignment in Cellular Networks. In: proc. of the 3rd International Conference on Algorithmic Aspects in Information and Management (AAIM 2007), pp. 191–200 (2007)
4. Caragiannis, I., Kaklamanis, C., Papaioannou, E.: Efficient on-line frequency allocation and call control in cellular networks. Theory Comput. Syst. 35(5), 521–543 (2002) A preliminary version of the paper appeared in SPAA 2000
5. Erlebach, T., Jacob, R., Mihalak, M., Nunkesser, M., Szabo, G., Widmayer, P.: An algorithmic view on OVSF code assignment. In: Diekert, V., Habib, M. (eds.) STACS 2004. LNCS, vol. 2996, pp. 270–281. Springer, Heidelberg (2004)
6. Forišek, M., Katreniak, B., Katreniaková, J., Královič, R., Koutný, V., Pardubská, D., Plachetka, T., Rovan, B.: Online bandwidth allocation. In: proc. of the 16th Annual European Symposium on Algorithms (ESA 2007) (to appear, 2007)
7. Li, X.Y., Wan, P.J.: Theoretically Good Distributed CDMA/OVSF Code Assignment for Wireless Ad Hoc Networks. In: Wang, L. (ed.) COCOON 2005. LNCS, vol. 3595, pp. 126–135. Springer, Heidelberg (2005)
8. Knuth, D.E.: The Art of Computer Programming. Fundamental Algorithms, vol. 1. Addison-Wesley, Reading (1975)
9. Kalyanasundaram, B., Pruhs, K.: Speed is as powerful as clairvoyance. In: Proceedings of the 36th IEEE Symposium on Foundations of Computer Science, pp. 214–221 (1995)
10. McDiarmid, C., Reed, B.A.: Channel assignment and weighted coloring. Networks 36(2), 114–117 (2000)
11. Minn, T., Siu, K.Y.: Dynamic assignment of orthogonal variable-spreading factor codes in W-CDMA. IEEE Journal on Selected Areas in Communications 18(8), 1429–1440 (2000)
12. Rouskas, A.N., Skoutas, D.N.: OVSF codes assignment and reassignment at the forward link OFW-CDMA 3G systems. In: Proc. of the 13 th IEEE International Symposium on Personal, Indoor and Mobile Radio Communications
13. Erlebach, T., Jacob, R., Tomamichel, M.: Algorithmische Aspekte von OVSF Code Assignment mit Schwerpunkt auf Offline Code Assignment. Student thesis at ETH Zürich
14. Wan, P.J., Li, X.Y., Frieder, O.: OVSF-CDMA Code Assignment in Wireless Ad Hoc Networks. In: Proc. of DIAL M-POMC 2004 joint workshop on Foundations of mobile computing, pp. 92–101 (2004)

Average-Case Analysis of Online Topological Ordering

Deepak Ajwani and Tobias Friedrich

Max-Planck-Institut für Informatik, Saarbrücken, Germany

Abstract. Many applications like pointer analysis and incremental compilation require maintaining a topological ordering of the nodes of a directed acyclic graph (DAG) under dynamic updates. All known algorithms for this problem are either only analyzed for worst-case insertion sequences or only evaluated experimentally on random DAGs. We present the first average-case analysis of online topological ordering algorithms. We prove an expected runtime of $\mathcal{O}(n^2 \text{ polylog}(n))$ under insertion of the edges of a complete DAG in a random order for the algorithms of Alpern et al. (SODA, 1990), Katriel and Bodlaender (TALG, 2006), and Pearce and Kelly (JEA, 2006). This is much less than the best known worst-case bound $\mathcal{O}(n^{2.75})$ for this problem.

1 Introduction

There has been a growing interest in dynamic graph algorithms over the last two decades due to their applications in a variety of contexts including operating systems, information systems, network management, assembly planning, VLSI design and graphical applications. Typical dynamic graph algorithms maintain a certain property (e. g., connectivity information) of a graph that changes (a new edge inserted or an existing edge deleted) dynamically over time. An algorithm or a problem is called *fully dynamic* if both edge insertions and deletions are allowed, and it is called *partially dynamic* if only one (either only insertion or only deletion) is allowed. If only insertions are allowed, the partially dynamic algorithm is called incremental; if only deletions are allowed, it is called decremental. While a number of fully dynamic algorithms have been obtained for various properties on undirected graphs (see (9) and references therein), the design and analysis of fully dynamic algorithms for directed graphs has turned out to be much harder (e. g., (12, 23, 24, 25)). Much of the research on directed graphs is therefore concentrated on the design of partially dynamic algorithms instead (e. g., (3, 6, 13)). In this paper, we focus on the analysis of algorithms for maintaining a topological ordering of directed graphs in an incremental setting.

A topological order T of a directed graph $G = (V, E)$ (with $n := |V|$ and $m := |E|$) is a linear ordering of its nodes such that for all directed paths from $x \in V$ to $y \in V$ $(x \neq y)$, it holds that $T(x) < T(y)$. A directed graph has a topological ordering if and only if it is acyclic. There are well-known algorithms for computing the topological ordering of a directed acyclic graph (DAG) in

T. Tokuyama (Ed.): ISAAC 2007, LNCS 4835, pp. 464–475, 2007.

$\mathcal{O}(m+n)$ time in an offline setting (see e. g. (7)). In a fully dynamic setting, each time an edge is added or deleted from the DAG, we are required to update the bijective mapping T. In the online/incremental variant of this problem, the edges of the DAG are not known in advance but are inserted one at a time (no deletions allowed). As the topological order remains valid when removing edges, most algorithms for online topological ordering can also handle the fully dynamic setting. However, there are no good bounds known for the fully dynamic case. Most algorithms are only analyzed in the online setting.

Given an arbitrary sequence of edges, the online cycle detection problem is to discover the first edge which introduces a cycle. Till now, the best known algorithm for this problem involves maintaining an online topological order and returning the edge after which no valid topological order exists. Hence, results for online topological ordering also translate into results for the online cycle detection problem. Online topological ordering is required for incremental evaluation of computational circuits (2) and in incremental compilation (15, 17) where a dependency graph between modules is maintained to reduce the amount of recompilation performed when an update occurs. An application for online cycle detection is pointer analysis (20).

The naïve way of computing an online topological order each time from scratch with the offline algorithm takes $\mathcal{O}(m^2+mn)$ time. Marchetti-Spaccamela, Nanni, and Rohnert (16) gave an algorithm that can insert m edges in $\mathcal{O}(mn)$ time. Alpern, Hoover, Rosen, Sweeney, and Zadeck (AHRSZ) proposed an algorithm (2) which runs in $\mathcal{O}\big(|\rangle\hat{K}\langle| \log(|\rangle\hat{K}\langle|)\big)$ time per edge insertion with $|\rangle\hat{K}\langle|$ being a local measure of the insertion complexity. However, there is no analysis of AHRSZ for a sequence of edge insertions. Katriel and Bodlaender (KB) (13) analyzed a variant of the AHRSZ algorithm and obtained an upper bound of $\mathcal{O}(\min\{m^{\frac{3}{2}}\log n, m^{\frac{3}{2}}+n^2\log n\})$ for inserting an arbitrary sequence of m edges. The algorithm by Pearce and Kelly (PK) (18) empirically outperforms the other algorithms for random edge insertions leading to sparse random DAGs, although its worst-case runtime is inferior to KB. Recently, Ajwani, Friedrich, and Meyer (AFM) (1) proposed a new algorithm with runtime $\mathcal{O}(n^{2.75})$, which asymptotically outperforms KB on dense DAGs.

As noted above, the empirical performances on random edge insertion sequences (REIS) for the above algorithms are quite different from their worst-cases. While PK performs empirically better for REIS, KB and AFM are the best known algorithms for worst-case sequences. This leads us to the theoretical study of online topological ordering algorithms on REIS.

Our contributions are as follows:

- We show an expected runtime of $\mathcal{O}(n^2\log^2 n)$ for inserting all edges of a complete DAG in a random order with PK (cf. Section 4).
- For AHRSZ and KB, we show an expected runtime of $\mathcal{O}(n^2\log^3 n)$ for complete random edge insertion sequences (cf. Section 5). This is significantly better than the known worst-case bound of $\mathcal{O}(n^3)$ for KB to insert $\Omega(n^2)$ edges.

- Additionally, we show that for such edge insertions the expected number of edges which force any algorithm to change the topological order ("invalidating edges") is $\mathcal{O}(n^{\frac{3}{2}}\sqrt{\log n})$ (cf. Section 6), which is the first such result.

The remainder of this paper is organized as follows. The next section describes briefly the three algorithms AHRSZ, KB, and PK. In Section 3 we specify the random graph models used in our analysis. Sections 4-6 prove our upper bounds for the runtime of the three algorithms and the number of invalidating edges.

2 Algorithms

This section first introduces some notations and then describes the three algorithms AHRSZ, KB, and PK. We keep the current topological order as a bijective function $T\colon V \to [1..n]$. In this and the subsequent sections, we will use the following notations: $d(u,v)$ denotes $|T(u) - T(v)|$, $u < v$ is a short form of $T(u) < T(v)$, $u \to v$ denotes an edge from u to v, and $u \rightsquigarrow v$ expresses that v is reachable from u. Note that $u \rightsquigarrow u$, but *not* $u \to u$. The *degree* of a node is the sum of its in- and out-degree.

Consider the i-th edge insertion $u \to v$. We say that an edge insertion is *invalidating* if $u > v$ before the insertion of this edge. We define $R_B^{(i)} := \{x \in V \mid v \leq x \wedge x \rightsquigarrow u\}$, $R_F^{(i)} := \{y \in V \mid y \leq u \wedge v \rightsquigarrow y\}$ and $\delta^{(i)} = R_F^{(i)} \cup R_B^{(i)}$. Let $|\delta^{(i)}|$ denote the number of nodes in $\delta^{(i)}$ and let $\|\delta^{(i)}\|$ denote the number of edges incident to any node of $\delta^{(i)}$. Note that $\delta^{(i)}$ as defined above is different from the adaptive parameter δ of the bounded incremental computation model. If an edge is non-invalidating, then $|R_B^{(i)}| = |R_F^{(i)}| = |\delta^{(i)}| = 0$. Note that for an invalidating edge $R_F^{(i)} \cap R_B^{(i)} = \emptyset$ as otherwise the algorithms will just report a cycle and terminate.

We now describe the insertion of the i-th edge $u \to v$ for all the three algorithms. Assume for the remainder of this section that $u \to v$ is an invalidating edge, as otherwise none of the algorithms do anything for that edge. We define an algorithm to be *local* if it only changes the ordering of nodes x with $v \leq x \leq u$ to compute the new topological order T' of $G \cup \{(u,v)\}$. All three algorithms are local and they work in two phases - a "discovery phase" and a "relabelling phase".

In the discovery phase of **PK**, the set $\delta^{(i)}$ is identified using a forward depth-first search from v (giving a set $R_F^{(i)}$) and a backward depth-first search from u (giving a set $R_B^{(i)}$). The relabelling phase is also very simple. They sort both sets $R_F^{(i)}$ and $R_B^{(i)}$ separately in increasing topological order and then allocate new priorities according to the relative position in the sequence $R_B^{(i)}$ followed by $R_F^{(i)}$. They do not alter the priority of any node not in $\delta^{(i)}$, thereby greatly simplifying the relabeling phase. The runtime of PK for a single edge insertion is $\Theta(\|\delta^{(i)}\| + |\delta^{(i)}| \log |\delta^{(i)}|)$.

Alpern et al. (2) used the bounded incremental computation model (23) and introduced the measure $|\rangle\hat{K}\langle|$. For an invalidated topological order T, the set

$K \subseteq V$ is a *cover* if for all $x, y \in V$: $(x \rightsquigarrow y \ \wedge \ y < x \ \Rightarrow \ x \in K \vee y \in K)$. This states that for any connected x and y which are incorrectly ordered, a cover K must include x or y or both. $|K|$ and $\|K\|$ denote the number of nodes and edges touching nodes in K, respectively. We define $|\rangle K \langle| := |K| + \|K\|$ and a cover $\hat{K}$ to be *minimal* if $|\rangle \hat{K} \langle| \leq |\rangle K \langle|$ for any other cover K. Thus, $|\rangle \hat{K} \langle|$ captures the minimal amount of work required to calculate the new topological order T' of $G \cup \{(u, v)\}$ assuming that the algorithm is local and that the adjacent edges must be traversed.

AHRSZs *discovery phase* marks the nodes of a cover K by marking some of the unmarked nodes $x, y \in \delta^{(i)}$ with $x \rightsquigarrow y$ and $y < x$. This is done recursively by moving two frontiers starting from v and u towards each other. Here, the crucial decision is which frontier to move next. AHRSZ tries to minimize $\|K\|$ by balancing the number of edges seen on both sides of the frontier. The recursion stops when forward and backward frontier meet. Note that we do not necessarily visit all nodes in $R_F^{(i)}$ ($R_B^{(i)}$) while extending the forward frontier (backward frontier). It can be proven that the marked nodes indeed form a cover K and that $|\rangle K \langle| \leq 3\,|\rangle \hat{K} \langle|$.

The *relabeling phase* employs the dynamic priority space data structure due to Dietz and Sleator (8). This permits new priorities to be created between existing ones in $\mathcal{O}(1)$ amortized time. This is done in two passes over the nodes in K. During the first pass, it visits the nodes of K in reverse topological order and computes a strict upper bound on the new priorities to be assigned to each node. In the second phase, it visits the nodes in K in topological order and computes a strict lower bound on the new priorities. Both together allow to assign new priorities to each node in K. Thereafter they minimize the number of different labels used to speed up the operations on the priority space data structure in practice. It can be proven that the discovery phase with $|\rangle \hat{K} \langle|$ priority queue operations dominates the time complexity, giving an overall bound of $\mathcal{O}(|\rangle \hat{K} \langle| \log |\rangle \hat{K} \langle|)$.

KB is a slight modification of AHRSZ. In the discovery phase AHRSZ counts the total number of edges incident on a node. KB counts instead only the in-degree of the backward frontier nodes and only the out-degree of the forward frontier nodes. In addition, KB also simplified the relabeling phase. The nodes visited during the extension of the forward (backward) frontier are deleted from the dynamic priority space data-structure and are reinserted, in the same relative order among themselves, after (before) all nodes in $R_B^{(i)}$ ($R_F^{(i)}$) not visited during the backward (forward) frontier extension. The algorithm thus computes a cover $K \subseteq \delta^{(i)}$ and its complexity per edge insertion is $\mathcal{O}(|\rangle K \langle| \log |\rangle K \langle|)$. The worst case running time of KB for a sequence of m edge insertions is $\mathcal{O}(\min\{m^{\frac{3}{2}} \log n, m^{\frac{3}{2}} + n^2 \log n\})$.

3 Random Graph Model

Erdős and Rényi (10, 11) introduced and popularized random graphs. They defined two closely related models: $G(n, p)$ and $G(n, M)$. The $G(n, p)$ model ($0 < p < 1$) consists of a graph with n nodes in which each edge is chosen

independently with probability p. On the other hand, the $G(n, M)$ model assigns equal probability to all graphs with n nodes and exactly M edges. Each such graph occurs with a probability of $1/\binom{N}{M}$, where $N := \binom{n}{2}$.

For our study of online topological ordering algorithms, we use the random DAG model of Barak and Erdős (4). They obtain a random DAG by directing the edges of an undirected random graph from lower to higher indexed vertices. Depending on the underlying random graph model, this defines the $DAG(n, p)$ and $DAG(n, M)$ model. We will mainly work on the $DAG(n, M)$ model since it is better suited to describe incremental addition of edges.

The set of all DAGs with n nodes is denoted by DAG^n. For a random variable f with probability space DAG^n, $\mathbf{E}_M(f)$ and $\mathbf{E}_p(f)$ denotes the expected value in the $DAG(n, M)$ and $DAG(n, p)$ model, respectively. For the remainder of this paper, we set $\mathbf{E}(f) := \mathbf{E}_M(f)$ and $q := 1 - p$.

The following theorem shows that in most investigations the models $DAG(n, p)$ and $DAG(n, M)$ are practically interchangeable, provided M is close to pN.

Theorem 1. *Given a function $f\colon DAG^n \to [0, a]$ with $a > 0$ and $f(G) \leq f(H)$ for all $G \subseteq H$ and functions p and M of n with $0 < p < 1$ and $M \in \mathbb{N}$.*

1. *If* $\lim_{n\to\infty} pqN = \lim_{n\to\infty} \dfrac{pN - M}{\sqrt{pqN}} = \infty$, *then* $\mathbf{E}_M(f) \leq \mathbf{E}_p(f) + o(1)$.
2. *If* $\lim_{n\to\infty} pqN = \lim_{n\to\infty} \dfrac{M - pN}{\sqrt{pqN}} = \infty$, *then* $\mathbf{E}_p(f) \leq \mathbf{E}_M(f) + o(1)$.

The analogous theorem for the undirected graph models $G(n, p)$ and $G(n, M)$ is well known. A closer look at the proof for it given by Bollobás (5) reveals that the probabilistic argument used to show the close connection between $G(n, p)$ and $G(n, M)$ can be applied in the same manner for the two random DAG models $DAG(n, p)$ and $DAG(n, M)$.

We define a random edge sequence to be a uniform random permutation of the edges of a complete DAG, i. e., all permutations of $\binom{n}{2}$ edges are equally likely. If the edges appear to the online algorithm in the order in which they appear in the random edge sequence, we call it a random edge insertion sequence (REIS). Note that a DAG obtained after inserting M edges of a REIS will have the same probability distribution as $DAG(n, M)$. To simplify the proofs, we first show our results in $DAG(n, p)$ model and then transfer them in the $DAG(n, M)$ model by Theorem 1.

4 Analysis of PK

When inserting the i-th edge $u \to v$, PK only regards nodes in $\delta^{(i)} := \{x \in V \mid v \leq x \leq u \wedge (v \rightsquigarrow x \vee x \rightsquigarrow u)\}$ with "$\leq$" defined according to the current topological order. As discussed in Section 2, PK performs $\mathcal{O}(\|\delta^{(i)}\| + |\delta^{(i)}| \log |\delta^{(i)}|)$ operations for inserting the i-th edge. Theorems 4 and 10 of this section show for a random edge insertion sequence (REIS) of N edges that $\sum_{i=1}^{N} |\delta^{(i)}| = \mathcal{O}(n^2)$ and $\mathbf{E}\big(\sum_{i=1}^{N} \|\delta^{(i)}\|\big) = \mathcal{O}(n^2 \log^2 n)$. This proves the following theorem.

Theorem 2. *For a random edge insertion sequence (REIS) leading to a complete DAG, the expected runtime of PK is* $\mathcal{O}(n^2 \log^2 n)$.

A comparable pair (of nodes) are two distinct nodes x and y such that either $x \rightsquigarrow y$ or $y \rightsquigarrow x$. We define a potential function Φ_i similar to Katriel and Bodlaender (13). Let Φ_i be the number of comparable pairs after the insertion of i edges. Clearly,

$$\begin{aligned} \Delta\Phi_i := \Phi_i - \Phi_{i-1} \geq 0 \quad &\text{for all } 1 \leq i \leq M, \\ \Phi_0 = 0, \quad &\text{and} \quad \Phi_M \leq n(n-1)/2. \end{aligned} \tag{1}$$

Theorem 3. *For all edge sequences, (i)* $|\delta^{(i)}| \leq \Delta\Phi_i + 1$ *and (ii)* $|\delta^{(i)}| \leq 2\Delta\Phi_i$.

Proof. Consider the i-th edge (u, v). If $u < v$, the theorem is trivial since $|\delta^{(i)}| = 0$. Otherwise, each vertex of $R_F^{(i)}$ and $R_B^{(i)}$ (as defined in Section 2) gets newly ordered with respect to u and v, respectively. The set $\bigcup_{x \in R_B^{(i)}}(x, v) \cap \bigcup_{x \in R_F^{(i)}}(u, x) = \{(u, v)\}$ as otherwise it will imply a discovered cycle and the algorithm will report the cycle and terminate. This means that overall at least $|R_F^{(i)}| + |R_B^{(i)}| - 1$ node pairs get newly ordered:

$$\Delta\Phi_i \geq |R_F^{(i)}| + |R_B^{(i)}| - 1 = |\delta^{(i)}| - 1.$$

Also, since in this case $\Delta\Phi_i \geq 1$, $|\delta^{(i)}| \leq 2\Delta\Phi_i$. □

Theorem 4. *For all edge sequences,* $\sum_{i=1}^{N} |\delta^{(i)}| \leq n(n-1) = \mathcal{O}(n^2)$.

Proof. By Theorem 3 (i), we get $\sum_{i=1}^{N} |\delta^{(i)}| \leq \sum_{i=1}^{N} (\Delta\Phi_i + 1) = \Phi_N + N \leq n(n-1)/2 + n(n-1)/2 = n(n-1)$. □

The remainder of this section provides the necessary tools step by step to finally prove the desired bound on $\sum_{i=1}^{N} \|\delta^{(i)}\|$ in Theorem 10. One can also interpret Φ_i as a random variable in $DAG(n, M)$ with $M = i$. The corresponding function Ψ for $DAG(n, p)$ is defined as the total number of comparable node pairs in $DAG(n, p)$. Pittel and Tungol (21) showed the following theorem.

Theorem 5. *For* $p := c \log(n)/n$ *and* $c > 1$, $\mathbf{E}_p(\Psi) = (1 + o(1)) \frac{n^2}{2} \left(1 - \frac{1}{c}\right)^2$.

Using Theorem 1, this result can be transformed to Φ as defined above for $DAG(n, M)$ and gives the following bounds for $\mathbf{E}_M(\Phi_k)$.

Theorem 6. *For* $n \log n < k \leq N - 2n \log n$,

$$\mathbf{E}_M(\Phi_k) = (1 + o(1)) \frac{n^2}{2} \left(1 - \frac{(n-1)\log n}{2(k + n \log n)}\right)^2.$$

For $N - 2n \log n < k \leq N - 2 \log n$,

$$\mathbf{E}_M(\Phi_k) = (1 + o(1)) \frac{n^2}{2} \left(1 - \frac{(n-1)\log n}{2(k + \sqrt{n\,(N-k)})}\right)^2.$$

The formal proof of the above theorem will be given in the full version of the paper.

The degree sequence of a random graph is a well-studied problem. The following theorem is shown in (5).

Theorem 7. *If $pn/\log n \to \infty$, then almost every graph G in the $G(n,p)$ model satisfies $\Delta(G) = (1+o(1))\,pn$, where $\Delta(G)$ is the maximum degree of a node in G.*

As noted in Section 3, the undirected graph obtained by ignoring the directions of $DAG(n,p)$ is a $G(n,p)$ graph. Therefore, the above result is also true for the maximum degree (in-degree + out-degree) of a node in $DAG(n,p)$. Using Theorem 1, the above result can be transformed to $DAG(n,M)$, as well.

Theorem 8. *With probability $1-\mathcal{O}(1/n)$, there is no node with degree higher than $c\frac{M}{n}$ for $n \geq n_0$ and $M > n\log n$ in $DAG(n,M)$, where c and n_0 are fixed constants.*

The formal proof for $c = 9$ will be given in the full version of the paper.

As the maximum degree of a node in $DAG(n,i)$ is $\mathcal{O}(i/n)$, we finally just need to show a bound on $\sum_i (i \cdot |\delta^{(i)}|)$ to prove Theorem 10. This is done in the following theorem.

Theorem 9. *For $DAG(n,M)$ and $r := N - 2\log n$,*

$$\mathbf{E}\Big(\sum_{i=1}^{r} (i \cdot |\delta^{(i)}|)\Big) = \mathcal{O}(n^3 \log^2 n).$$

Proof. Let us decompose the analysis in three steps. First, we show a bound on the first $n\log n$ edges. By definition of $\delta^{(i)}$, $|\delta^{(i)}| \leq n$. Therefore,

$$\sum_{i=1}^{n\log n} i \cdot \mathbf{E}(|\delta^{(i)}|) \leq \sum_{i=1}^{n\log n} i \cdot n = \mathcal{O}\left(n^3 \log^2 n\right). \tag{2}$$

The second step is to bound $\sum_{i=n\log n}^{t} i \cdot |\delta^{(i)}|$ with $t := N - 2n\log n$. For this, Theorem 3 (ii) shows for all k such that $n\log n < k < t$ that

$$\mathbf{E}\Big(\sum_{i=k}^{t} |\delta^{(i)}|\Big) \leq 2\,\mathbf{E}\Big(\sum_{i=k}^{t} \Delta\Phi_i\Big) = 2\,\mathbf{E}(\Phi_t - \Phi_{k-1}) = 2\,\mathbf{E}(\Phi_t) - 2\mathbf{E}(\Phi_{k-1}).$$

Using Theorem 6 and the fact that the hidden functions of the $o(1)$ are decreasing in p (21), this yields (with $s := n\log n$)

$$\begin{aligned}
\mathbf{E}\Big(\sum_{i=k}^{t} |\delta^{(i)}|\Big) &\leq (1+o(1))\,n^2\left(\left(1-\frac{(n-1)\log n}{2(t+s)}\right)^2 - \left(1-\frac{(n-1)\log n}{2(k-1+s)}\right)^2\right) \\
&= (1+o(1))\,n^2(n-1)\log n\Big(\frac{2}{2(k-1+s)} - \frac{2}{2(t+s)} + \\
&\qquad \frac{(n-1)\log n}{4}\Big(\frac{1}{(t+s)^2} - \frac{1}{(k-1+s)^2}\Big)\Big) \\
&\leq (1+o(1))\,n^2(n-1)\log n\left(\frac{1}{k-1+s} - \frac{1}{t+s}\right) \\
&\leq (1+o(1))\,n^2(n-1)\log n\frac{1}{k-1}.
\end{aligned} \tag{3}$$

By linearity of expectation and Equation (3),

$$\begin{aligned}
\mathbf{E}\Big(\sum_{i=s+1}^{t} i\,|\delta^{(i)}|\Big) &= \sum_{i=s+1}^{t}\Big(i\,\mathbf{E}(|\delta^{(i)}|)\Big) \le \sum_{j=1}^{\log(\lceil \frac{t}{s}\rceil)}\Big(2^j s \sum_{i=2^{(j-1)}s+1}^{2^j s}\mathbf{E}(|\delta^{(i)}|)\Big)\\
&\le \sum_{j=1}^{\log(\lceil \frac{t}{s}\rceil)}\Big(2^j s \sum_{i=2^{(j-1)}s+1}^{t}\mathbf{E}(|\delta^{(i)}|)\Big)\\
&\le \sum_{j=1}^{\log(\lceil \frac{t}{s}\rceil)}\Big(2^j s(1+o(1))\,n^2(n-1)\log n\frac{1}{2^{(j-1)}s}\Big)\\
&= \sum_{j=1}^{\log(\lceil \frac{t}{s}\rceil)}\Big(2(1+o(1))\,n^2(n-1)\log n\Big)\\
&= 2(1+o(1))\,n^2(n-1)\log^2 n = \mathcal{O}(n^3\log^2 n).
\end{aligned}$$

For the last step consider a k such that $t < k < r$. Theorem 3 (ii) gives

$$\mathbf{E}\Big(\sum_{i=k}^{r}|\delta^{(i)}|\Big) \le 2\,\mathbf{E}\Big(\sum_{i=k}^{r}\Delta\Phi_i\Big) = 2\,\mathbf{E}(\Phi_r-\Phi_{k-1}) = 2\,\mathbf{E}(\Phi_r) - 2\mathbf{E}(\Phi_{k-1}).$$

Using Theorem 6 and similar arguments as before, this yields (with $s(k) := \sqrt{\log n\ (N-k)}$)

$$\begin{aligned}
\mathbf{E}\Big(\sum_{i=k}^{r}|\delta^{(i)}|\Big) &\le (1+o(1))\,n^2\left(\Big(1-\frac{(n-1)\log n}{2(r+s(r))}\Big)^2-\Big(1-\frac{(n-1)\log n}{2(k-1+s(k-1))}\Big)^2\right)\\
&= (1+o(1))\,n^2(n-1)\log n\left(\frac{2}{2(k-1+s(k-1))}-\frac{2}{2(r+s(r))}+\right.\\
&\qquad\left.\frac{(n-1)\log n}{4}\Big(\frac{1}{(r+s(r))^2}-\frac{1}{(k-1+s(k-1))^2}\Big)\right).
\end{aligned}$$

Since $k+s(k)$ is monotonically increasing for $t < k < r$, $\frac{1}{(k+s(k))^2}$ is a monotonically decreasing function in this interval. Therefore, $\frac{1}{(r+s(r))^2}-\frac{1}{(k-1+s(k-1))^2} < 0$, which proves the following equation.

$$\begin{aligned}
\mathbf{E}\Big(\sum_{i=k}^{r}|\delta^{(i)}|\Big) &\le (1+o(1))\,n^2(n-1)\log n\left(\frac{1}{k-1+s(k-1)}-\frac{1}{r+s(r)}\right)\\
&\le (1+o(1))\,n^2(n-1)\log n\frac{1}{k-1}. \qquad (4)
\end{aligned}$$

By linearity of expectation and Equation (4),

$$\begin{aligned}
\mathbf{E}\Big(\sum_{i=N-2n\log n+1}^{r} |\ ^{(i)}|\Big) &= \sum_{i=N-2n\log n+1}^{r} \Big(\ \mathbf{E}(|\ ^{(i)}|)\Big) \\
&\le (\quad -2\log\ \ ) \sum_{i=N-2n\log n+1}^{r} \mathbf{E}(|\ ^{(i)}|) \\
&\le (\quad -2\log\ \ )(1+\ \ (1))\ \ ^{2}(\ \ -1)\log\ \ \frac{1}{-2\ \ \log\ \ -1} \\
&= \mathcal{O}(\ \ ^{3}\log\ \ )
\end{aligned}$$

□

Theorem 10. *For DAG*(n, M), $\mathbf{E}\Big(\sum_{i=1}^{N} \|\delta^{(i)}\|\Big) = \mathcal{O}(n^2 \log^2 n)$.

Proof. By definition of $\|\delta^{(i)}\|$, we know $\|\delta^{(i)}\| \le i$ and hence

$$\sum_{i=1}^{n\log n} \|\delta^{(i)}\| = \mathcal{O}(n^2 \log^2 n).$$

Again, let $r := N - 2\log n$. Theorem 8 tells us that with probability greater than $\left(1 - \frac{c'}{n}\right)$ for some constant c', there is no node with degree $\ge \frac{c\ i}{n}$. Since the degree of an arbitrary node in a DAG is bounded by n, we get with Theorems 4 and 9,

$$\begin{aligned}
\mathbf{E}\Big(\sum_{i=n\log n+1}^{r} \|\delta^{(i)}\|\Big) &= \mathcal{O}\left(\mathbf{E}\Big(\sum_{i=n\log n+1}^{r} \frac{c\ i\ |\delta^{(i)}|}{n}\Big) + \mathbf{E}\Big(\sum_{i=n\log n+1}^{r} \frac{n\ c'\ |\delta^{(i)}|}{n}\Big)\right) \\
&= \mathcal{O}\Big(\frac{1}{n}\,\mathbf{E}\Big(\sum_{i=1}^{r} (i\ |\delta^{(i)}|)\Big) + n^2\Big) \\
&= \mathcal{O}\Big(\frac{1}{n}\left(n^3 \log^2 n\right) + n^2\Big) = \mathcal{O}(n^2 \log^2 n).
\end{aligned}$$

By again using the fact that the degree of an arbitrary node in a DAG is at most n, we obtain

$$\mathbf{E}\Big(\sum_{i=r+1}^{N} \|\delta^{(i)}\|\Big) = \mathcal{O}\Big(n \cdot \mathbf{E}\Big(\sum_{i=r+1}^{N} |\delta^{(i)}|\Big)\Big) = \mathcal{O}\Big(n \cdot \sum_{i=r+1}^{N} n\Big) = \mathcal{O}(n^2 \log n).$$

Thus,

$$\begin{aligned}
\mathbf{E}\Big(\sum_{i=1}^{N} \|\delta^{(i)}\|\Big) &= \mathbf{E}\Big(\sum_{i=1}^{n\log n} \|\delta^{(i)}\|\Big) + \mathbf{E}\Big(\sum_{i=n\ \log n+1}^{r} \|\delta^{(i)}\|\Big) + \mathbf{E}\Big(\sum_{i=r+1}^{N} \|\delta^{(i)}\|\Big) \\
&= \mathcal{O}(n^2 \log^2 n) + \mathcal{O}(n^2 \log^2 n) + \mathcal{O}(n^2 \log n) = \mathcal{O}(n^2\ \log^2 n).
\end{aligned}$$

□

5 Analysis of AHRSZ and KB

Katriel and Bodlaender (13) introduced KB as a variant of AHRSZ for which a worst-case runtime of $\mathcal{O}(\min\{m^{\frac{3}{2}}\log n, m^{\frac{3}{2}}+n^2\log n\})$ can be shown. In this section, we prove an expected runtime of $\mathcal{O}(n^2\log^3 n)$ under random edge insertion sequences, both for AHRSZ and KB.

Recall from Section 2 that for every edge insertion there is a minimal cover $\hat{K}^{(i)}$. In appendix C, we show that $\delta^{(i)}$ is also a valid cover in this situation. Therefore, by definition of $|\rangle\hat{K}^{(i)}\langle|$, $|\rangle\hat{K}^{(i)}\langle| \le |\rangle\delta^{(i)}\langle| = |\delta^{(i)}| + \|\delta^{(i)}\|$.

$$\mathbf{E}\Big(\sum_{i=1}^{m}|\rangle\hat{K}^{(i)}\langle|\Big) \le \sum_{i=1}^{m}|\delta^{(i)}| + \mathbf{E}\Big(\sum_{i=1}^{m}\|\delta^{(i)}\|\Big) = \mathcal{O}(n^2\log^2 n)$$

The latter equality follows from Theorems 4 and 10. The expected complexity of AHRSZ on REIS is thus $\mathcal{O}\big(\mathbf{E}\big(\sum_{i=1}^{m}|\rangle\hat{K}^{(i)}\langle|\log n\big)\big) = \mathcal{O}(n^2\log^3 n)$.

KB also computes a cover $K \subseteq \delta^{(i)}$ and its complexity per edge insertion is $\mathcal{O}(|\rangle K\langle|\,\log|\rangle K\langle|)$. Therefore, $|\rangle K\langle| \le |\delta^{(i)}|+\|\delta^{(i)}\|$ and with a similar argument as above, the expected complexity of KB on REIS is $\mathcal{O}(n^2\log^3 n)$.

6 Bounding the Number of Invalidating Edges

An interesting question in all this analysis is how many edges will actually invalidate the topological ordering and force any algorithm to do something about them. Here, we show a non-trivial upper bound on the expected value of the number of invalidating edges on REIS. Consider the following random variable: $\textsc{inval}(i) = 1$ if the i-th edge inserted is an invalidating edge; $\textsc{inval}(i) = 0$ otherwise.

Theorem 11. $\mathbf{E}\Big(\sum_{i=1}^{m}\textsc{inval}(i)\Big) = \mathcal{O}(\min\{m, n^{\frac{3}{2}}\log^{\frac{1}{2}} n\})$.

Proof. If the i-th edge is invalidating, $|\delta^{(i)}| \ge 2$; otherwise $\textsc{inval}(i) = |\delta^{(i)}| = 0$. In either case, $\textsc{inval}(i) \le |\delta^{(i)}|/2$. Thus, for $s := n^{\frac{3}{2}}\log^{\frac{1}{2}} n$ and $t := \min\{m, N - 2n\log n\}$,

$$\mathbf{E}\Big(\sum_{i=s+1}^{t}\textsc{inval}(i)\Big) \le \mathbf{E}\Big(\sum_{i=s+1}^{t}\frac{|\delta^{(i)}|}{2}\Big) \le \frac{(1+o(1))}{2}\,n^{\frac{3}{2}}\log^{\frac{1}{2}} n.$$

The second inequality follows by substituting $k := s+1$ in Equation (3). Also, since the number of invalidating edges can be at most equal to the total number of edges, $\sum_{i=1}^{s}\textsc{inval}(i) \le s$.

$$\begin{aligned}\mathbf{E}\Big(\sum_{i=1}^{m}\textsc{inval}(i)\Big) &= \mathbf{E}\Big(\sum_{i=1}^{s}\textsc{inval}(i)\Big) + \mathbf{E}\Big(\sum_{i=s+1}^{t}\textsc{inval}(i)\Big) + \mathbf{E}\Big(\sum_{i=t}^{m}\textsc{inval}(i)\Big)\\ &\le \mathcal{O}(s) + \mathcal{O}(n^{\frac{3}{2}}\log^{\frac{1}{2}} n) + \mathcal{O}(n\log n) = \mathcal{O}(n^{\frac{3}{2}}\log^{\frac{1}{2}} n).\end{aligned}$$

The second bound $\mathbf{E}\,(\sum_{i=1}^{m}\textsc{inval}(i)) \le m$ is obvious by definition of $\textsc{inval}(i)$. □

7 Discussion

On random edge insertion sequences (REIS) leading to a complete DAG, we have shown an expected runtime of $\mathcal{O}(n^2 \log^2 n)$ for PK and $\mathcal{O}(n^2 \log^3 n)$ for AHRSZ and KB while the trivial lower bound is $\Omega(n^2)$. Extending the average case analysis for the case where we only insert m edges with $m \ll n^2$ still remains open. On the other hand, the only non-trivial lower bound for this problem is by Ramalingam and Reps (22), who have shown that an adversary can force any algorithm which maintains explicit labels to require $\Omega(n \log n)$ time complexity for inserting $n-1$ edges. There is still a large gap between the lower bound of $\Omega(\max\{n \log n, m\})$, the best average-case bound of $\mathcal{O}(n^2 \log^2 n)$ and the worst-case bound of $\mathcal{O}(\min\{m^{1.5} + n^2 \log n, m^{1.5} \log n, n^{2.75}\})$. Bridging this gap remains an open problem.

Acknowledgements

The authors are grateful to Telikepalli Kavitha, Irit Katriel, and Ulrich Meyer for various helpful discussions.

References

[1] Ajwani, D., Friedrich, T., Meyer, U.: An $O(n^{2.75})$ algorithm for online topological ordering. In: Arge, L., Freivalds, R. (eds.) SWAT 2006. LNCS, vol. 4059, pp. 53–64. Springer, Heidelberg (2006)
[2] Alpern, B., Hoover, R., Rosen, B.K., Sweeney, P.F., Zadeck, F.K.: Incremental evaluation of computational circuits. In: Proc. of SODA 1990, pp. 32–42 (1990)
[3] Ausiello, G., Italiano, G.F., Marchetti-Spaccamela, A., Nanni, U.: Incremental algorithms for minimal length paths. J. Algorithms 12(4), 615–638 (1991)
[4] Barak, A.B., Erdős, P.: On the maximal number of strongly independent vertices in a random acyclic directed graph. SIAM Journal on Algebraic and Discrete Methods 5(4), 508–514 (1984)
[5] Bollobás, B.: Random Graphs. Cambridge Univ. Press, Cambridge (2001)
[6] Cicerone, S., Frigioni, D., Nanni, U., Pugliese, F.: A uniform approach to semi-dynamic problems on digraphs. Theor. Comput. Sci. 203(1), 69–90 (1998)
[7] Cormen, T., Leiserson, C., Rivest, R.: Introduction to Algorithms. MIT Press, Cambridge (1989)
[8] Dietz, P.F., Sleator, D.D.: Two algorithms for maintaining order in a list. In: Proc. of STOC 1987, pp. 365–372 (1987)
[9] Eppstein, D., Galil, Z., Italiano, G.F.: Dynamic graph algorithms. In: Atallah, M.J. (ed.) Algorithms and Theory of Computation Handbook. ch. 8, CRC Press (1999)
[10] Erdős, P., Rényi, A.: On random graphs. Publ. Math. Debrecen. 6, 290–297 (1959)
[11] Erdős, P., Rényi, A.: On the evolution of random graphs. Magyar Tud. Akad. Mat. Kutato Int. Kozl 5, 17–61 (1960)
[12] Frigioni, D., Marchetti-Spaccamela, A., Nanni, U.: Fully dynamic shortest paths and negative cycles detection on digraphs with arbitrary arc weights. In: Bilardi, G., Pietracaprina, A., Italiano, G.F., Pucci, G. (eds.) ESA 1998. LNCS, vol. 1461, pp. 320–331. Springer, Heidelberg (1998)

[13] Katriel, I., Bodlaender, H.L.: Online topological ordering. ACM Trans. Algorithms 2(3), 364–379 (2006) Preliminary version appeared as [14]
[14] Katriel, I., Bodlaender, H.L.: Online topological ordering. In: Proc. of SODA 2005, pp. 443–450 (2005)
[15] Marchetti-Spaccamela, A., Nanni, U., Rohnert, H.: On-line graph algorithms for incremental compilation. In: van Leeuwen, J. (ed.) WG 1993. LNCS, vol. 790, pp. 70–86. Springer, Heidelberg (1994)
[16] Marchetti-Spaccamela, A., Nanni, U., Rohnert, H.: Maintaining a topological order under edge insertions. Information Processing Letters 59(1), 53–58 (1996)
[17] Omohundro, S.M., Lim, C.-C., Bilmes, J.: The sather language compiler/debugger implementation. Technical Report 92-017, International Computer Science Institute, Berkeley (1992)
[18] Pearce, D.J., Kelly, P.H.J.: A dynamic topological sort algorithm for directed acyclic graphs. J. Exp. Algorithmics 11(1.7) (2006) Preliminary version appeared as [19]
[19] Pearce, D.J., Kelly, P.H.J.: A dynamic algorithm for topologically sorting directed acyclic graphs. In: Ribeiro, C.C., Martins, S.L. (eds.) WEA 2004. LNCS, vol. 3059, pp. 383–398. Springer, Heidelberg (2004)
[20] Pearce, D.J., Kelly, P.H.J., Hankin, C.: Online cycle detection and difference propagation: Applications to pointer analysis. Software Quality Journal 12(4), 311–337 (2004)
[21] Pittel, B., Tungol, R.: A phase transition phenomenon in a random directed acyclic graph. Random Struct. Algorithms 18(2), 164–184 (2001)
[22] Ramalingam, G., Reps, T.W.: On competitive on-line algorithms for the dynamic priority-ordering problem. Information Processing Letters 51, 155–161 (1994)
[23] Ramalingam, G., Reps, T.W.: On the computational complexity of dynamic graph problems. Theor. Comput. Sci. 158(1-2), 233–277 (1996)
[24] Roditty, L., Zwick, U.: A fully dynamic reachability algorithm for directed graphs with an almost linear update time. In: Proc. of STOC 2004, pp. 184–191 (2004)
[25] Roditty, L., Zwick, U.: On dynamic shortest paths problems. In: Albers, S., Radzik, T. (eds.) ESA 2004. LNCS, vol. 3221, pp. 580–591. Springer, Heidelberg (2004)

Energy Efficient Deadline Scheduling in Two Processor Systems

Tak-Wah Lam[1], Lap-Kei Lee[1], Isaac K. K. To[2,⋆], and Prudence W. H. Wong[2,⋆]

[1] Department of Computer Science, University of Hong Kong, Hong Kong
{twlam,lklee}@cs.hku.hk
[2] Department of Computer Science, University of Liverpool, UK
{isaacto,p.wong}@liverpool.ac.uk

Abstract. The past few years have witnessed different scheduling algorithms for a processor that can manage its energy usage by scaling dynamically its speed. In this paper we attempt to extend such work to the two-processor setting. Specifically, we focus on deadline scheduling and study online algorithms for two processors with an objective of maximizing the throughput, while using the smallest possible energy. The motivation comes from the fact that dual-core processors are getting common nowadays. Our first result is a new analysis of the energy usage of the speed function OA [15,4,8] with respect to the optimal two-processor schedule. This immediately implies a trivial two-processor algorithm that is 16-competitive for throughput and $O(1)$-competitive for energy. A more interesting result is a new online strategy for selecting jobs for the two processors. Together with OA, it improves the competitive ratio for throughput from 16 to 3, while increasing that for energy by a factor of 2. Note that even if the energy usage is not a concern, no algorithm can be better than 2-competitive with respect to throughput.

1 Introduction

Energy usage is an important concern in the design of modern processors. To be more energy efficient, many modern processors adopt the technology of dynamic speed scaling, where the processor can adjust its speed dynamically in some range without any overhead. In general, running a processor at speed s would consume energy at the rate s^α, where α is a constant believed to be 2 or 3 (see, e.g., [7,9,13,14]). That is, a processor can save energy by running slower. This energy saving capability has triggered a lot of work to revisit processor scheduling; the concern is how to exploit this capability to reduce the energy usage, while achieving as much as possible the original scheduling objectives (such as throughput and flow time).

The pioneering work along this direction was due to Yao et al. [15]. They considered the problem of deadline scheduling on a processor that can vary

⋆ This research is partly supported by EPSRC Grant EP/E028276/1.

T. Tokuyama (Ed.): ISAAC 2007, LNCS 4835, pp. 476–487, 2007.

its speed between 0 and infinity. We call this the infinite speed model. In this model, it is always feasible to complete all jobs by their deadlines; the concern is how to adjust the processor speed and whether the total energy usage can be $O(1)$-competitive. Yao et al. answered in the affirmative by showing an online algorithm called AVR to be $2^{\alpha-1}\alpha^\alpha$-competitive, and they proposed another algorithm called OA (Optimal Available), which is later proved by Bansal et al. [4] to be α^α-competitive. Bansal et al. also gave a new algorithm that is $2(\alpha/(\alpha-1))^\alpha e^\alpha$ competitive. Note that all these algorithms only give a speed function for the processor; all jobs are scheduled in an EDF (Earliest Deadline First) manner. Recently several interesting results on flow time scheduling have also been revealed under the infinite speed model (see [1,5,10]).

The infinite speed model is a convenient model for studying different speed functions and their energy usage. Yet in reality, the speed of a processor is bounded. Chan et al. [8] recently initiated the study of the bounded speed model, where a processor can vary its speed between 0 and a fixed maximum speed T. In this model, deadline scheduling becomes more complicated and even the optimal offline algorithm may not be able to complete all jobs. A natural objective is to maximize the throughput (i.e., the total work of jobs that can be completed by their deadlines), while using the smallest possible energy. They showed that the energy demanded by the speed function OAT, which is simply OA capped at T, is at most $\alpha^\alpha + 4^\alpha\alpha^2$ times of the energy of any offline algorithm that produces the maximum throughput. They further showed that OAT can support a job selection strategy called FSA to be 14-competitive for throughput. More recently, Bansal et al. [3] showed that OAT is indeed "fast" enough to support a more aggressive job selection strategy called Slow-D, thus improving the competitive ratio from 14 to 4. Note that even if the energy issue is ignored, no algorithm can be better than 4-competitive with respect to throughput [6].

Many modern processors are indeed dual-core (or even quad-core) processors. It is natural to extend the study of energy efficient scheduling to the multi-processor setting. In this paper we take the first step to investigate the case of two processors, and we would like to derive an online algorithm with a throughput that can match or is close to the known lower bound, while being $O(1)$-competitive on energy. In the infinite speed model, it is relatively trivial to derive a two-processor algorithm that can complete all jobs (1-competitive for throughput) and is $O(1)$-competitive for energy (see explanation below). It is the bounded speed model that needs attention. In this model, if energy is not a concern, it has been known that the two-processor algorithm Safe-Risky is 2-competitive for throughput [11], and no online algorithm can be better than 2-competitive [12]. Thus, it is natural to ask for an online algorithm for two processors that is 2 or close to 2 competitive for throughput and $O(1)$-competitive for energy.

To ease our discussion, we let Opt_1 to denote the optimal offline algorithm for a single processor that maximizes the throughput, while using the smallest energy, and similarly Opt_2 for the case of two processors. Furthermore, let $E_{\text{Opt}i}$ and $W_{\text{Opt}i}$ denote respectively the energy usage and throughput of Opt_i, where

$i = 1$ or 2. First of all, let us explain why scheduling two processors is relatively trivial in the infinite speed model. When the processor speed is unbounded, all jobs can be completed on time even using one single processor. Furthermore, we can use a single processor, which runs sufficiently fast, to simulate any two-processor schedule that completes all jobs; the energy usage increases by at most $2^{\alpha-1}$ times. Thus, $E_{\text{Opt1}} \leq 2^{\alpha-1} E_{\text{Opt2}}$. Given two processors, a trivial algorithm is to use OA for one processor and let the other idle. This would give an algorithm that is 1-competitive for throughput and $2^{\alpha-1}\alpha^\alpha$-competitive for energy.

The rest of this paper is devoted to the bounded speed model. First of all, let us look at the performance of the trivial algorithm that uses Slow-D and OAT on one processor, leaving the other idle. The simulation argument above is no longer valid in the bounded speed model, yet it is probably still true that $E_{\text{Opt1}} \leq 2^{\alpha-1} E_{\text{Opt2}}$, implying that the energy usage of OAT (and Slow-D) is at most $(2^{\alpha-1}\alpha^\alpha + 2^{3\alpha-1}\alpha^2) E_{\text{Opt2}}$. In this paper we give a better analysis of OAT, showing that the energy usage of OAT is at most

$$(2^{\alpha-1}\alpha^\alpha + 2^{2\alpha-1}\alpha^2) E_{\text{Opt2}}.$$

For the throughput, we can easily argue that $W_{\text{Opt1}} \geq W_{\text{Opt2}}/4$ (using the notion of minimally infeasible job set given in [8]). Thus, the trivial two-processor algorithm is 16-competitive for throughput and $O(1)$-competitive for energy.

The above competitive ratio for throughput is far from the lower bound of 2. This motivates us to develop a non-trivial strategy to schedule jobs with both processors. We develop a new job selection strategy called Slow-SR, with one processor following a schedule similar to Slow-D and the other working like the Risky processor of the Safe-Risky algorithm. The competitive ratio of throughput is reduced from 16 to 3. Both processors are running at a speed not exceeding that of OAT and the competitive ratio for energy becomes $2^\alpha\alpha^\alpha + 4^\alpha\alpha^2$.

In conclusion, we have devised a two-processor algorithm for the bounded speed model, which is 3-competitive for throughput and $2^\alpha\alpha^\alpha + 4^\alpha\alpha^2$-competitive for energy. As to be explained, our algorithm requires job migration from the Risky processor to the Slow-D processor. It is interesting to find a non-migratory algorithm with similar performance (though in a dual-core processor, the two CPUs share the same memory, so the overhead of migration is not as expensive as the typical distributed model). We believe that for $m \geq 2$ processors, it is possible to have an algorithm that is $O(1)$-competitive for both throughput and energy usage. Yet it is hard to generalize the algorithm given in this paper. On the other hand, Albers et al. [2] have recently obtained some interesting results on multiprocessor deadline scheduling for the infinite speed model. They focused on the special case where jobs have agreeable deadlines (i.e., the deadlines of the jobs follow the order of their release times), and they showed a non-migratory multiprocessor algorithm that is $\alpha^\alpha 16^\alpha$-competitive for energy (recall that in the infinite speed model, the competitive ratio of throughput is always one).

Organization of the paper. In Section 2, we describe the algorithm Slow-SR. In Section 3, we establish a couple of key properties of the algorithm. Using

these properties, we analyze the throughput of Slow-SR in Section 4. Finally, we analyze the energy competitiveness of OAT in Section 5.

Definitions and Assumptions. Before we give the details of Slow-SR, let us define the scheduling problem formally. There are two processors, the speed of each can be independently scaled from 0 to a maximum speed T. A processor running at speed s can process s units of work per unit of time, consuming energy at the rate s^α. Jobs are released in an online fashion. Each job j is characterized by a release time r_j, a deadline d_j, and required work (or size) p_j. At any time, a job can be executed in only one of the two processors. Preemption and migration is allowed with no penalty. A job j is said to be completed if p_j units of work has been processed by its deadline. Note that there may be too many jobs to be completed. The objective of a scheduler is to maximize the throughput, while minimizing the energy. An online algorithm is said to be c-competitive for throughput and c'-competitive for energy if, for any job sequence, its throughput is at least $\frac{1}{c}W_{\text{Opt2}}$ and its energy usage is at most $c'E_{\text{Opt2}}$. To simplify our discussion, we assume that some suitable scaling has been done to the processor speed and job size so that $T = 1$.[1]

2 The Slow-SR Algorithm

The algorithm Slow-SR makes reference to the schedule of OA. Before describing the algorithm Slow-SR, we need a review of OA.

OA works for one processor only. We characterize OA by the schedule it plans to use at any time t, assuming no more jobs are released after t. Let $I(t)$ be the subset of jobs that has arrived up to time t. We use $\rho(t_1, t_2)$ to denote the remaining work of the jobs in $I(t)$ with deadlines in the interval $(t_1, t_2]$. The speed function that OA plans to use looks like a staircase, with speed reduced at certain "critical" times c_0, c_1, ... defined as follows. Let $c_0 = t$. For any i, the speed that OA plans to use immediately after c_i is $\rho_{i+1} = \max_{t' > c_i} \rho(c_i, t')/(t' - c_i)$; OA maintains this speed until c_{i+1}, defined as the earliest time after c_i such that $\rho(c_i, c_{i+1})/(c_{i+1} - c_i) = \rho_{i+1}$. The intervals $[c_i, c_{i+1}]$ are called the critical intervals. Within each critical interval, jobs with deadlines in this interval are executed in an EDF order. Intuitively, OA is very lazy; within each critical interval, OA just uses the minimum speed that would not cause any job to miss a deadline. Note that if no more jobs are released, OA never changes the speed planned as above.

Consider any time t. With respect to the speed function planned by OA at time t, we define $t_{\text{slow}} \geq t$ to be the first time when OA plans to use speed 1 or less. Furthermore, we say that t is a "slow time" if OA actually runs at speed 1

[1] Given a job set I to be scheduled on a processor with maximum speed $T > 1$, we define another job set I' by scaling the work of each job j in I to p_j/T. Then any schedule for I' with maximum speed 1 can be transformed to a schedule for I with maximum speed T (by increasing the speed by a factor of T), and vice versa. Therefore, the competitive ratios for throughput and energy both preserve.

Data Structures:
Q_{slow}, initially $\emptyset$
Q_{fast}, initially $\emptyset$
J_{uc}, initially nil
Scheduling framework:
At fast time, SP runs earliest deadline job in Q_{fast} at speed 1
At slow time, SP runs earliest deadline job in Q_{slow} at the same speed as OA
If $J_{\text{uc}} \neq nil$, RP runs J_{uc} at speed 1
Event: Release of job j
1: Update the simulated OA schedule by adding j
2: Move jobs in Q_{slow} with deadline at or before t_{slow} to Q_{fast}
3: **if** $d_j > t_{\text{slow}}$ **then**
4: Add j to Q_{slow}
5: **else if** $\{j\} \cup Q_{\text{fast}}$ is feasible **then**
6: Add j to Q_{fast}
Event: LST interrupt of job j
7: **if** $J_{\text{uc}} = nil$ or $p(j) > p(J_{\text{uc}})$ **then**
8: $J_{\text{uc}} \leftarrow j$
Event: Job completion in SP
9: Remove the completed job from Q_{slow} or Q_{fast}
10: **if** $J_{\text{uc}} \neq nil$ and $Q_{\text{fast}} = \emptyset$ **then**
11: $Q_{\text{fast}} \leftarrow \{J_{\text{uc}}\}$
12: $J_{\text{uc}} \leftarrow nil$

Algorithm 1. Slow-SR

or less at t, and a "fast time" otherwise. By definition, OA plans to change speed at t_{slow}, and t_{slow} is a critical time. That means, OA plans to execute only jobs with deadlines at or before t_{slow} until t_{slow}.

Slow-SR is defined in Algorithm 1.The two processors are labeled as SP (Slow-D Processor) and RP (Risky Processor), respectively. The algorithm keeps two queues (sets), both on jobs committed to run on SP. The queue Q_{slow} contains jobs that OA plans to run "slowly" (at speed 1 or less), and the queue Q_{fast} for jobs that OA plans to run "quickly" (at speed over 1). The algorithm also keeps a job J_{uc} committed to run on RP. For each job j not in Q_{fast} or Q_{slow}, an LST (latest-start-time) interrupt will occur at time $d_j - p_j$, which attempts to retain job j as J_{uc} for execution.

3 Relations of the Job Queues

We establish two main characteristics of Slow-SR in this section, namely that SP completes all jobs that have ever entered Q_{fast} and Q_{slow}, and that RP is busy only during fast time. For any time t, let $t_{\text{slow}}(t)$ denotes t_{slow} at t, before events at t (if any) are taken into account. A job j is said to be *active* at time t if $t < d_j$ and j has not yet been completed by t. Before we begin, it is essential for us to understand the following implications of slow time.

Lemma 1. *At any slow time, all active jobs are in* Q_{slow}.

Proof. Let t be a slow time. So $t_{\text{slow}}(t) = t$. For any active job j, $d_j > t = t_{\text{slow}}(t) \geq t_{\text{slow}}(t')$ for any $t' \in [r_j, t]$, where the last inequality follows from the nature of OA that $t_{\text{slow}}(t)$ increases monotonically with t. Thus j entered Q_{slow} at release, and is not moved to Q_{fast} at or before t. So all active jobs are in Q_{slow}. □

Lemma 2. *The schedules of OA and Slow-SR during slow time are exactly the same (assuming they break ties of deadlines in the same way).*

Proof. Since Slow-SR uses the speed of OA during slow time, we only need to consider the choices of jobs of the two schedulers. Note that during slow time, both schedulers always choose the earliest deadline active job (for Slow-SR this is because Lemma 1 guarantees that Q_{slow} contains all active jobs). So it suffices to show that they have the same set of active jobs. Let t be the first slow time that OA and Slow-SR have different set of active jobs, and thus can have different scheduling. Let j be any job active at t in either scheduler, we will show that it is active in both schedulers, meaning the active job sets are the same, i.e., contradiction. We just need to observe that neither scheduler runs j during fast time before t, for OA it is because $d_j > t = t_{\text{slow}}(t) \geq t_{\text{slow}}(t')$ for any $t' < t$, so it has too late deadline; and for Slow-SR it is because the above implies that j has never been in Q_{fast}. In other words, scheduling of j before t must be exactly the same in the two schedulers, so it must be active in both. □

We now prove the two main properties of Slow-SR.

Property 1. SP completes all jobs that have ever entered Q_{fast} or Q_{slow}.

Proof. For jobs that are added to Q_{slow} at release and are never moved to Q_{fast}, Lemma 2 guarantees that their scheduling is the same as OA and will be completed. We now show that Q_{fast} remains feasible (using a speed 1 processor) at any time, which implies that each job in Q_{fast} is feasible at their deadlines, i.e., already completed at their deadlines. We first check that Q_{fast} does not become infeasible without jobs being added to it: if Q_{fast} is non-empty but feasible at t, there must be active jobs in Q_{fast}, and Lemma 1 implies that it is fast time. SP thus runs the earliest deadline job in Q_{fast} using speed 1, so Q_{fast} is feasible immediately after t.

Now we turn to the cases when jobs enter Q_{fast}. For cases where there is a feasibility check, the feasibility after the job entering Q_{fast} is trivial. The only remaining case is when jobs are moved from Q_{slow} to Q_{fast}. If Q_{fast} is feasible before such a move, then $Q_{\text{fast}} \cup Q_{\text{slow}}$ must also be feasible before such a move since their time of execution do not conflict: jobs in Q_{fast} have deadlines at or before $t_{\text{slow}}(t)$ and thus can only be processed at or before $t_{\text{slow}}(t)$, while scheduling of Q_{slow} is always the same as OA (Lemma 1) which plans to use time after $t_{\text{slow}}(t)$ to work on Q_{slow}. It is then obvious that the move does not cause infeasibility. □

Property 2. If $J_{\text{uc}} \neq \mathit{nil}$ at t, then (1) $Q_{\text{fast}} \neq \emptyset$, and (2) t must be a fast time.

Proof. Suppose at time t, $J_{\text{uc}} = j$ and $Q_{\text{fast}} = \emptyset$. So j must have caused an LST interrupt at some time $t_l < t$, and remains as J_{uc} since then. If Q_{fast} is non-empty at t_l, there must be a time in (t_l, t) at which Q_{fast} changes from non-empty to empty, when j would be moved to Q_{fast}, contradicting that $J_{\text{uc}} = j$ at t. But if Q_{fast} is empty at t_l, it means j can be completed with all jobs completed in Q_{fast} by t_l, so j should enter Q_{fast} at r_j, contradicting that J caused an LST interrupt. Therefore, $Q_{\text{fast}} \neq \emptyset$. This also implies that some active job is in Q_{fast} (Property 1), so by Lemma 1, the current time is a fast time. □

4 Throughput Competitiveness

We analyze the throughput of Slow-SR in this section. This is done by partitioning the set of jobs into two categories: L_s includes those that enter Q_{slow} on release, and L_f includes all other jobs. The jobs in L_s are all completed by Slow-SR (Property 1), while Slow-SR can miss the deadlines of some jobs in L_f. Note that the spans (i.e., time interval between release time and deadline) of jobs in L_f are completely in fast time, putting a bound on the amount of work that the optimal algorithm can complete for L_f. Let f denote the total length of periods of fast time. The core of the proof is Lemma 5, allowing us to show that Slow-SR completes f units of work, resulting into the following theorem.

Theorem 1. *Slow-SR is 3-competitive on throughput.*

Proof. The optimal offline schedule can at most complete all jobs in L_s and $2f$ units of work in L_f, leading to a total throughput of $W_{\text{Opt}} \leq p(L_s) + 2f$. Slow-SR completes all jobs in L_s, so its throughput $W_{\text{Slow-SR}} \geq p(L_s)$. We will show in Lemma 3 that for jobs with deadlines in each maximal period of fast time F, Slow-SR completes an amount of work no less than the length of F. Summing over all such maximal periods, it means for jobs with deadlines in fast time, Slow-SR completes no less than f units of work, leading to $W_{\text{Slow-SR}} \geq f$. In conclusion, $W_{\text{Opt}} \leq W_{\text{Slow-SR}} + 2W_{\text{Slow-SR}} \leq 3W_{\text{Slow-SR}}$, completing the proof. □

It is tricky to show that Slow-SR completes enough work during each maximal period of fast time F. Although all jobs in SP meet deadlines (Property 1), and SP always works at speed 1 during fast time, Slow-SR can be idle in some fast time. So we consider each busy period within each maximal period of fast time. Let t_0, t_1, ..., t_l be the list of times in F from earliest to latest when one of the following happens: (1) F begins, (2) F ends, or (3) SP switches from idle to busy. So $F = [t_0, t_l]$. Note that some job runs in SP at t_0, since some jobs with deadlines no later than t_{slow} must be released at the beginning of any fast time period, so if Q_{fast} is not occupied by some previously released jobs, it must accept one of those newly released jobs.

Lemma 3. *For jobs with deadlines in a maximal period of fast time F, Slow-SR completes an amount of work no less than the length of F.*

Proof. In Lemma 5, we will show that the amount of work completed before t_n for jobs with deadlines in $(t_{\text{slow}}(t_{n-1}), t_{\text{slow}}(t_n)]$, plus the amount of work completed during $[t_{n-1}, t_n]$ for jobs with deadlines in $(t_{n-1}, t_{\text{slow}}(t_{n-1})]$, is at least $t_{\text{slow}}(t_n) - t_{\text{slow}}(t_{n-1})$. Now sum over all n from 1 to l. We can check that no work completed by Slow-SR is being counted more than once, and all of them are jobs with deadlines in F. So the work completed for jobs with deadlines in F is at least $\sum_{n=1}^{l} t_{\text{slow}}(t_n) - t_{\text{slow}}(t_{n-1}) = t_{\text{slow}}(t_l) - t_{\text{slow}}(t_0) = t_l - t_0$, i.e., the length of F. □

To establish Lemma 5, we depend on two facts. The first is that whenever a job j cannot enter either Q_{fast} or Q_{slow} because a job j' in $Q_{\text{fast}} \cup \{j\}$ would miss deadline, SP and RP of Slow-SR must be able to process an amount of useful work no less than $d_{j'} - r_j$. We say j' *repudiates* j in such cases. The second is a property of OA concerning $t_{\text{slow}}(t)$: the amount of work in jobs already released at t with deadlines between $t' \geq t$ and $t_{\text{slow}}(t)$ cannot be small.

Lemma 4. *Let t be a fast time. For any $t' \in [t, t_{\text{slow}}(t))$, the amount of work in jobs with deadlines in $(t', t_{\text{slow}}(t)]$ released before t is more than $t_{\text{slow}}(t) - t'$.*

Proof. Immediately before t, consider the jobs that OA may plan to run during $[t', t_{\text{slow}}(t)]$. It cannot run jobs with deadlines at or before t', because these jobs have deadlines passed. By the nature of OA, it does not plan to run jobs with deadlines after $t_{\text{slow}}(t)$ until $t_{\text{slow}}(t)$. So it can only plan to run jobs with deadlines $[t', t_{\text{slow}}(t)]$. Yet it plans to use faster than speed 1 during that whole period (so that the period is fast). The lemma arrives immediately. □

Lemma 5. *Consider any $n \in \{1, \ldots, l\}$. The amount of work completed before t_n for jobs with deadlines in $(t_{\text{slow}}(t_{n-1}), t_{\text{slow}}(t_n)]$, plus the amount of work completed during $[t_{n-1}, t_n]$ for jobs with deadlines in $(t_{n-1}, t_{\text{slow}}(t_{n-1})]$, is at least $t_{\text{slow}}(t_n) - t_{\text{slow}}(t_{n-1})$.*

Proof. For $t \in [t_{n-1}, t_n]$, let $P(t)$ be the following proposition: If all jobs with release time at or after t are not released, the amount of work completed by Slow-SR before $t_{\text{slow}}(t)$ for jobs with deadlines in $(t_{\text{slow}}(t_{n-1}), t_{\text{slow}}(t)]$, plus the amount of work completed by Slow-SR during $[t_{n-1}, t_{\text{slow}}(t)]$ for jobs with deadlines in $(t_{n-1}, t_{\text{slow}}(t_{n-1})]$, is no less than $t_{\text{slow}}(t) - t_{\text{slow}}(t_{n-1})$.

When t is set to be the last busy time t_b in $[t_{n-1}, t_n]$, this is exactly Lemma 5: First, the work that would be done before $t_{\text{slow}}(t_b)$ if no more jobs are released at or after t_b is exactly the same as the amount of work actually completed before t_b (since Slow-SR idles afterwards). Second, $t_{\text{slow}}(t_b)$ is the same as $t_{\text{slow}}(t_n)$ since it cannot change between t_b and t_n, without causing a job to be accepted into Q_{fast} and extending the current busy period or starting a new one.

The base case $t = t_{n-1}$ is trivial. Suppose $P(t)$ is true for all $t \in [t_{n-1}, u)$. We now establish $P(u)$. It only uses release times less than t, so the transfinite induction works (i.e., it always terminates after a finite number of steps).

Let S be the set of jobs with deadlines in $(t_{\text{slow}}(t_{n-1}), t_{\text{slow}}(u)]$ already released before u. By Lemma 4, the amount of work in S must be more than $t_{\text{slow}}(u) - t_{\text{slow}}(t_{n-1})$. If all jobs in S enter Q_{fast} or Q_{slow} on release, $P(u)$ is satisfied

by just these jobs. We thus assume that some jobs in S are not accepted into Q_{fast} or Q_{slow} on release, i.e., some repudiation occurred during $[t_{n-1}, u)$. Let j_r be the latest deadline repudiating job during these repudiations, which occurred at $t_r < t$. The set of jobs Γ with deadlines in $(d_{j_r}, t_{\text{slow}}(u)]$ released before u must all be accepted into Q_{fast} or Q_{slow} on release, since no job can repudiate them. This also shows that d_{j_r} must be larger than $t_{\text{slow}}(t_{n-1})$, otherwise all jobs with deadlines in $(t_{\text{slow}}(t_{n-1}), t_{\text{slow}}(t_n)]$ must be admitted into Q_{fast} or Q_{slow}, contradicting our assumption.

If no job in Γ is run at any fast time before r_{j_r}, we can check that the total completed work counted in $P(u)$ is at least $t_{\text{slow}}(u) - t_{n-1} > t_{\text{slow}}(u) - t_{\text{slow}}(t_{n-1})$, so $P(u)$ is satisfied. The amount of work $t_{\text{slow}}(u) - t_{n-1}$ comes from three disjoint parts of useful work processed by SP and RP: (1) Those processed before $t_{\text{slow}}(u)$ with deadlines in $(d_{j_r}, t_{\text{slow}}(u)]$—at least $t_{\text{slow}}(u) - d_{j_r}$ by Lemma 4; (2) Those processed by SP during $[t_{n-1}, t_r]$—exactly $t_r - t_{n-1}$; and (3) Those with deadlines at or before d_{j_r} that are planned to be processed by SP at the repudiation time t_r, plus the job that is being repudiated or the replacement job that eventually get completed by RP (with or without the help of SP)—the repudiation implies this to be at least $d_{j_r} - t_r$.

Finally, we consider the case if some job $j_e \in \Gamma$ runs at some fast time before r_{j_r}. Let t_e be a time immediately after such execution. Note that $t_e \geq t_{n-1}$: otherwise j_e must be in Q_{fast} and partially executed immediately before t_{n-1}, contradicting that SP is slow or idle by then. By $P(t_e)$ (induction hypothesis), the amount of work counted by $P(t_e)$ is no less than $t_{\text{slow}}(t_e) - t_{\text{slow}}(t_{n-1})$. To see $P(u)$ is satisfied, we note that $P(u)$ includes the work completed for jobs with deadlines in $(t_{\text{slow}}(t_e), t_{\text{slow}}(u)]$, which is not counted in $P(t_e)$. This is more than $t_{\text{slow}}(u) - t_{\text{slow}}(t_e)$ by Lemma 4, so the amount of work counted by $P(u)$ is more than $t_{\text{slow}}(t_e) - t_{\text{slow}}(t_{n-1}) + t_{\text{slow}}(u) - t_{\text{slow}}(t_e) = t_{\text{slow}}(u) - t_{\text{slow}}(t_{n-1})$. □

5 Energy Competitiveness

We analyze the energy consumption of the single processor schedule OA capped at a maximum speed of 1 (OAT) in this section, showing that it is $(2^{\alpha-1}\alpha^\alpha + 2^{2\alpha-1}\alpha^2)$-competitive in energy against the minimum energy 2-processor offline schedule that achieves the maximum throughput. By Property 2, the speeds of both processors of Slow-SR never exceed that of OAT, so this implies that Slow-SR is $(2^\alpha\alpha^\alpha + 2^{2\alpha}\alpha^2)$-competitive in energy against this offline schedule. The competitive ratio of OAT also implies that Slow-D [3] is $(2^{\alpha-1}\alpha^\alpha + 2^{2\alpha-1}\alpha^2)$-competitive in energy in the same setting.

The analysis of OAT against optimal 2-processor schedule is a modification of the proof presented in [8]. Let's review the notations used. At any time t, we use $E_{\text{OAT}}(t)$ and $E_{\text{Opt}}(t)$ to denote the amount of energy already spent by OAT and OPT respectively. We overload the speed function OAT with an actual schedule that always executes the same job as OA, using the OAT speed function (i.e., capped at speed 1). This way, OAT always processes a job whenever its speed is non-zero, but may not process enough work to complete some jobs. In contrast,

OPT never processes a job without eventually completing it. We call work that is done for a job completed by OPT to be type-1 work, and other work to be type-0 work.

Consider OA at any time t. We use two functions $\phi(t)$ and $\beta(t)$ (as in [8]). The function $\phi(t)$ is 0 at the beginning and the end of the schedule, and changes continuously except when jobs are released. The function $\beta(t)$ is α^2 times the amount of type-0 work that would be completed by OAT if no more jobs are released after t. Our main theorem and the outline of its proof is as follows.

Theorem 2. $E_{\mathrm{OAT}} \leq (2^{\alpha-1}\alpha^{\alpha} + 2^{2\alpha-1}\alpha^2)E_{\mathrm{Opt}}$.

Proof. We will show that $E_{\mathrm{OAT}}(t)+\phi(t)-\beta(t) \leq 2^{\alpha-1}\alpha^{\alpha}E_{\mathrm{Opt}}(t)$. Before showing how to prove the inequality, let's consider its consequence. Consider a time t_e after all job deadlines. At that time, $E_{\mathrm{OAT}}(t_e) = E_{\mathrm{OAT}}$, $E_{\mathrm{Opt}}(t_e) = E_{\mathrm{Opt}}$, and $\phi(t_e) = 0$. So $E_{\mathrm{OAT}} - \beta(t_e) \leq 2^{\alpha-1}\alpha^{\alpha}E_{\mathrm{Opt}}$. Lemma 6 will show that $\beta(t_e) \leq 2^{2\alpha-1}\alpha^2 E_{\mathrm{Opt}}$. The theorem arrives immediately.

We now prove that $E_{\mathrm{OAT}}(t) + \phi(t) - \beta(t) \leq 2^{\alpha-1}\alpha^{\alpha}E_{\mathrm{Opt}}(t)$. Before any job is released, the inequality holds trivially, since all the terms are 0. Lemma 7 will show that when no job is being released, the rate of changes of the terms in the inequality satisfies $E_{\mathrm{OAT}}(t)'+\phi(t)'-\beta(t)' \leq 2^{\alpha-1}\alpha^{\alpha}E_{\mathrm{Opt}}(t)'$. Lemma 8 will show that when a job is released, the change of the terms in the inequality satisfies $\Delta\phi(t) - \Delta\beta(t) \leq 0$. $E_{\mathrm{OAT}}(t)$ and $E_{\mathrm{Opt}}(t)$ obviously remain unchanged. Thus no event invalidates the inequality since the time before any job is released. □

Now we bound the amount of type-0 work that OAT eventually completes.

Lemma 6. *If t_e is a time after the deadlines of all jobs, $\beta(t_e) \leq 2^{2\alpha-1}\alpha^2 E_{\mathrm{Opt}}$.*

Proof. We first bound E_{Opt}. Let S be a subset of I, the input job set. We say S is minimally infeasible if S is infeasible, but any proper subset of S is feasible, using a speed-1 processor. The union of spans of jobs in a minimally infeasible job set, $span(S)$, forms a continuous time interval (otherwise one of the sub-intervals alone must be infeasible). Let $\mathcal{M}$ be the set of all minimally infeasible subsets of I. Let $span(\mathcal{M})$ be the union of all spans of those subsets within $\mathcal{M}$. A job j is "overloaded" if its span is in $span(\mathcal{M})$ completely, and "underloaded" if otherwise. So all jobs in $\mathcal{M}$ are overloaded. In [8], it is shown that (1) all type-0 jobs are overloaded, and (2) underloaded jobs do not affect the feasibility of job sets. Because of the latter, OPT must maximize the amount of work completed in overloaded jobs.

Let $\mathcal{M}_0$ be a minimal subset of $\mathcal{M}$ with $span(\mathcal{M}_0) = span(\mathcal{M})$. The spans of minimally infeasible subsets in $\mathcal{M}_0$ must all have different start times in their spans, otherwise one of them can be removed from $\mathcal{M}_0$, so $\mathcal{M}_0$ is not minimal. So let $\mathcal{M}_0 = \{S_1, S_2, \ldots\}$, where the start time of span of S_1 is earlier than that of S_2, etc. The span of S_i cannot overlap with the span of S_{i+2} or any later subsets, since otherwise S_{i+1} can be removed from $\mathcal{M}_0$. It is thus feasible to complete using one processor all jobs in S_1, S_3, ... except the smallest job in each subset; and to complete using the other processor all jobs in S_2, S_4,

... except the smallest job in each subset, leading to an amount of work no less than half of the total length of $span(\mathcal{M})$ (which we will denote by $|span(\mathcal{M})|$). The minimum energy schedule for OPT to complete this amount of work within a time period of $|span(\mathcal{M})|$ is to spread it over all the time in two processors. This results in speed $1/4$ throughout the span, so $E_{\text{Opt}} \geq 2|span(\mathcal{M})|/4^\alpha$. Recall that $\beta(t_e)$ is α^2 times the amount of type-0 work completed by OAT. Since all type-0 work are overloaded, they must be executed in $span(\mathcal{M})$, resulting in $\beta(t_e) \leq \alpha^2|span(\mathcal{M})| \leq 2^{2\alpha-1}\alpha^2 E_{\text{Opt}}$ as claimed. □

To continue with the proof we need the definition of $\phi(t)$. Recall that the plan of OA defines critical times $c_0, c_1, \ldots$; and during each critical interval $[c_i, c_{i+1}]$, OA plans to use constant speed ρ_{i+1}. Let $w_{OAT}(i)$ be the amount of unfinished work under OAT in jobs with deadlines in $(c_{i-1}, c_i]$, according to the schedule if no more jobs are to be released after t. Let $w_{OPT}(i)$ be the amount of unfinished type-1 work under OPT in jobs with deadlines in $(c_{i-1}, c_i]$. Then

$$\phi(t) = \sum_{i \geq 1} (\min\{\rho_i, 1\})^{\alpha-1} (\alpha\, w_{OAT}(i) - \alpha^2 w_{OPT}(i)) \ .$$

We analyze the rate of change in terms of the inequality $E_{\text{OAT}}(t) + \phi(t) - \beta(t) \leq 2^{\alpha-1}\alpha^\alpha E_{\text{Opt}}(t)$, when no job is released.

Lemma 7. *When no job arrives,* $E_{\text{OAT}}(t)' + \phi(t)' - \beta(t)' - 2^{\alpha-1}\alpha^\alpha E_{\text{Opt}}(t)' \leq 0$.

Proof. Without new jobs, the expected schedule used by OAT is not changed, so $\beta(t)' = 0$. OAT runs at speed $s = \min\{\rho_1, 1\}$, so $E_{\text{OAT}}(t)' = s^\alpha$. For OPT, assume its two processors run at speed s_1 and s_2, so $E_{\text{Opt}}(t)' = s_1^\alpha + s_2^\alpha$. We need to bound $\phi(t)'$ from above. The function $\phi(t)$ consists of two sets of components, one for $w_{OAT}(i)$ and the other for $w_{OPT}(i)$. For $w_{OAT}(i)$, only $w_{OAT}(1)$ is changing, at a rate of $-s$. For $w_{OPT}(i)$, we do not know which i corresponds to the running jobs in the two processors, but $i = 1$ has the largest scaling factor, which results in the largest change of $\phi(t)$. So we have $\phi(t)' \leq s^{\alpha-1}(-\alpha s + \alpha^2(s_1 + s_2)) = -\alpha s^\alpha + \alpha^2 s^{\alpha-1}(s_1 + s_2)$. Therefore,

$$\begin{aligned}
&E_{\text{OAT}}(t)' + \phi(t)' - \beta(t)' - 2^{\alpha-1}\alpha^\alpha E_{\text{Opt}}(t)' \\
&\leq s^\alpha - \alpha s^\alpha + \alpha^2 s^{\alpha-1}(s_1 + s_2) - 2^{\alpha-1}\alpha^\alpha(s_1^\alpha + s_2^\alpha) \\
&= \frac{(1-\alpha)s^\alpha + 2\alpha^2 s^{\alpha-1}s_1 - 2^\alpha\alpha^\alpha s_1^\alpha}{2} + \frac{(1-\alpha)s^\alpha + 2\alpha^2 s^{\alpha-1}s_2 - 2^\alpha\alpha^\alpha s_2^\alpha}{2} \\
&= (s_1^\alpha f(s/s_1) + s_2^\alpha f(s/s_2))/2 \ ,
\end{aligned}$$

where $f(z) = (1-\alpha)z^\alpha + 2\alpha^2 z^{\alpha-1} - 2^\alpha\alpha^\alpha$. We note that $f(0) = -2^\alpha\alpha^\alpha < 0$, and when $z \to \infty$, $f(z) \to (1-\alpha)z^\alpha < 0$. For maximum value, we set $f'(z) = (1-\alpha)\alpha z^{\alpha-1} + 2\alpha^2(\alpha-1)z^{\alpha-2} = 0$, which implies that $z = 2\alpha$, and $f(z) = (1-\alpha)(2\alpha)^\alpha + 2\alpha^2(2\alpha)^{\alpha-1} - (2\alpha)^\alpha = 0$. So $f(z)$ is non-positive for any $z \geq 0$, concluding our proof. □

Finally, we note that the proof in [8] can be used directly to show the following lemma, which concerns with how the release of jobs (rather than the running of jobs) affects $\phi(t)$ and $\beta(t)$.

Lemma 8. *At the time when a job is released,* $\Delta\phi(t) - \Delta\beta(t) \leq 0$.

References

1. Albers, S., Fujiwara, H.: Energy-efficient algorithms for flow time minimization. In: Durand, B., Thomas, W. (eds.) STACS 2006. LNCS, vol. 3884, pp. 621–633. Springer, Heidelberg (2006)
2. Albers, S., Muller, F., Schmelzer, S.: Speed scaling on parallel processors. In: Proc. SPAA (2007)
3. Bansal, N., Chan, H.L., Lam, T.W., Lee, L.K.: Scheduling for speed bounded processors. Manuscript
4. Bansal, N., Kimbrel, T., Pruhs, K.: Dynamic speed scaling to manage energy and temperature. In: Proc. FOCS, pp. 520–529 (2004)
5. Bansal, N., Pruhs, K., Stein, C.: Speed scaling for weighted flow time. In: Proc. SODA, pp. 805–813 (2007)
6. Baruah, S., Koren, G., Mishra, B., Raghunathan, A., Rosier, L., Shasha, D.: On-line scheduling in the presence of overload. In: Proc. FOCS, pp. 100–110 (1991)
7. Brooks, D.M., Bose, P., Schuster, S.E., Jacobson, H., Kudva, P.N., Buyuktosunoglu, A., Wellman, J.D., Zyuban, V., Gupta, M., Cook, P.W.: Power-aware microarchitecture: Design and modeling challenges for next-generation microprocessors. IEEE Micro 20(6), 26–44 (2000)
8. Chan, H.L., Chan, W.T., Lam, T.W., Lee, L.K., Mak, K.S., Wong, P.: Energy efficient online deadline scheduling. In: Proc. SODA (2007)
9. Grunwald, D., Levis, P., Farkas, K.I., Morrey, C.B.: Policies for dynamic clock scheduling. In: Proc. OSDI, pp. 73–86 (2000)
10. Irani, S., Pruhs, K.R.: Algorithmic problems in power management. SIGACT News 36(2), 63–76 (2005)
11. Koren, G.: Competitive On-Line Scheduling for Overloaded Real-Time Systems. PhD thesis, New York University (1993)
12. Koren, G., Shasha, D.: MOCA: A multiprocessor on-line competitive algorithm for real-time system scheduling. Theor. Comput. Sci. 128(1&2), 75–97 (1994)
13. Pillai, P., Shin, K.G.: Real-time dynamic voltage scaling for low-power embedded operating systems. In: Proc. SOSP, pp. 89–102 (2001)
14. Weiser, M., Welch, B., Demers, A., Shenker, S.: Scheduling for reduced CPU energy. In: Proc. OSDI, pp. 13–23 (1994)
15. Yao, F., Demers, A., Shenker, S.: A scheduling model for reduced CPU energy. In: Proc. FOCS, pp. 374–382 (1995)

On the Relative Dominance of Paging Algorithms*

Reza Dorrigiv, Alejandro López-Ortiz, and J. Ian Munro

Cheriton School of Computer Science,
University of Waterloo,
Waterloo, ON, N2L 3G1, Canada
{rdorrigiv,alopez-o,imunro}@uwaterloo.ca

Abstract. In this paper we give a finer separation of several known paging algorithms. This is accomplished using a new technique that we call relative interval analysis. The technique compares the fault rate of two paging algorithms across the entire range of inputs of a given size rather than in the worst case alone. Using this technique we characterize the relative performance of LRU and LRU-2, as well as LRU and FWF, among others. We also show that lookahead is beneficial for a paging algorithm, a fact that is well known in practice but it was, until recently, not verified by theory.

1 Introduction

Paging is a fundamental problem in the context of the analysis of online algorithms. A paging algorithm mediates between a slower and a faster memory. Assuming a cache of size k, the algorithm decides which k memory pages to keep in the cache without the benefit of knowing in advance the *sequence* of upcoming page requests. After receiving the i^{th} page request the online algorithm must decide which page to evict, in the event the request results in a fault and the cache is full. The objective is to design efficient online algorithms in the sense that on a given request sequence the total cost, namely the total number of faults, is kept low. Three well known paging algorithms are *Least-Recently-Used* (LRU), *First-In-First-Out* (FIFO), and *Flush-When-Full* (FWF). On a fault, if the cache is full, LRU evicts the page that is least recently requested, FIFO evicts the page that is first brought to the cache, and FWF empties the cache.

The competitive ratio, first introduced formally by Sleator and Tarjan [20], has served as a practical measure for the study and classification of online algorithms. An algorithm (assuming a cost-minimization problem) is said to be α-competitive if the cost of serving any specific request sequence never exceeds α times the optimal cost (up to some additive constant) of an optimal *offline*

* This work was supported by grants from the Software Telecommunications Group at the University of Waterloo, NSERC of Canada and the Canada Research Chairs Program.

T. Tokuyama (Ed.): ISAAC 2007, LNCS 4835, pp. 488–499, 2007.

algorithm which knows the entire sequence. The competitive ratio has been applied to a variety of online problems and settings: is relatively simple measure to apply yet powerful enough to quantify, to a large extent, the performance of many an online algorithm. On the other hand, it has been observed by numerous researchers [5,7,14,21,10,17] that for paging it produces results that are too pessimistic or otherwise found wanting. For example, experimental studies show that LRU has a performance ratio at most four times the optimal offline algorithm [21,19], as opposed to the competitive ratio k predicted by competitive analysis. Furthermore, it has been empirically well established that LRU (and/or variants thereof) are, in practice, preferable paging strategies to all other known paging algorithms [18]. This is in contrast to competitive analysis in which the competitive ratio of LRU is the same as FWF and worse than some randomized algorithms. An additional drawback of competitive analysis, as can easily be shown [6], is that finite lookahead yields no improvement in the performance of an online algorithm. Once again, this is a rather counterintuitive conclusion: in practice, one expects that lookahead should improve performance, and a "reasonable" theoretical measure should reflect this "reality".

A careful study of the competitive ratio reveals the nature of the shortcomings. Chief among them are its focus on worst case behaviour and indirect comparison of online algorithms via an offline optimal algorithm. In the former case, this might lead, as observed above, to the competitive ratio declaring two wildly differing algorithms "equal" if they happen to err by the same amount in their worst possible input, even though in most other inputs one is superior to the other (e.g. LRU versus FWF). As well the indirect comparison via an offline optimal can introduce spurious artifacts from the comparison to an objects of a different type, namely an online to offline algorithm.

Such anomalies have lead to the introduction of several alternatives to competitive analysis of online algorithms, e.g., *loose competitiveness* [21,24], the *Max/Max Ratio* [5], *diffuse adversary* [14,22,23,4], the *random order ratio* [13], the *relative worst order ratio* [10,9,11], *access graph* [7], *concave analysis* [2], and *adequate analysis* [17]. We refer the reader to the survey of Dorrigiv and López-Ortiz [12] for a more comprehensive and detailed exposition. These measures achieve partial separation between well known algorithms for paging. Loose competitiveness, diffuse adversary model, and adequate analysis provide more realistic ratios for paging algorithms. The Max/Max ratio and the relative worst order ratio provide direct comparison between online algorithms and reflect the influence of lookahead. Recently, Angelopoulos et al. introduced *Bijective Analysis* and *Average Analysis* [3] which combined with the locality model of Albers et al. [2], shows that LRU is the sole optimal paging algorithm on sequences with locality of reference. This resolved an important disparity between theory and practice of online paging algorithms, namely the superiority in practice of LRU. A remaining question however, is how to characterize the full spectrum of performance of the various known paging algorithms. As discussed above, the competitive ratio focuses on the worst case which in this setting is known to be the same for all algorithms. In this paper we compare instead the performance

of two algorithms across the entire range of inputs; in that comparison we use the fault rate measure instead of the competitive ratio. Aside from artifacts introduced by the comparison to an offline algorithm, practitioners find the fault rate a better indicator of performance. Formally, the fault rate of a paging algorithm $\mathcal{A}$ on a sequence σ of length n is the number of faults that $\mathcal{A}$ incurs on $\mathcal{A}$ divided by n. The fault rate of $\mathcal{A}$ on a set of sequences is the worst (maximum) fault rate of $\mathcal{A}$ on any of those sequences. The idea behind the fault rate is that sequences on which $\mathcal{A}$ incurs very few faults compared to the number of requests are not that important, even if the number of faults happens to be much higher than what can be achieved by an offline (or even online) optimum. Consider the following example. Let $\mathcal{A}$ and $\mathcal{B}$ two online paging algorithms so that $\mathcal{A}$ incurs fewer faults than $\mathcal{B}$ on most sequences. Suppose that the fault rate of $\mathcal{A}$ is much lower than that of $\mathcal{B}$, so clearly $\mathcal{A}$ is preferable to $\mathcal{B}$. However, if there happens to be an "easy" sequence σ of length 1000000 on which $\mathcal{A}$ incurs 100 faults, $\mathcal{B}$ incurs 10 faults and optimal offline algorithm can serve σ by only 2 faults, then the competitive ratio of $\mathcal{A}$ is 50 while that of $\mathcal{B}$ is 5 suggesting the opposite of the logical conclusion. Note that the fault rates of $\mathcal{A}$ and $\mathcal{B}$ on σ are 0.01 and 0.001, respectively, which are miniscule and thus of no relevance in the actual performance of a system using either algorithm.

Our results: In this paper we aim to provide a tool for finer study and separation of the relative performance characteristics of online paging algorithms. We propose the *relative interval* approach which directly compares two online paging algorithms $\mathcal{A}$ and $\mathcal{B}$, without any reference to the optimal offline algorithm. They are compared across their entire performance spectrum (rather than on the worst case alone) using a normalized measure of performance, similar to the fault rate. Informally the relative interval of two algorithms reflects the range of the difference between the fault rate of those algorithms. For every two online paging algorithm $\mathcal{A}$ and $\mathcal{B}$ we define a relative interval $\mathcal{I}(\mathcal{A}, \mathcal{B}) = [\alpha, \beta]$, where $-1 \leq \alpha \leq 1$ and $-1 \leq \beta \leq 1$ (Note that if we consider "sensible" methods such that each is at least as good as ever other on at least some input then we have $-1 \leq \alpha \leq 0 \leq \beta \leq 1$). $\beta > -\alpha$, shows that $\mathcal{B}$ is better than $\mathcal{A}$ according to the relative interval. The more the difference, the better $\mathcal{B}$ is compared to $\mathcal{A}$. Table 1 shows the summary of our results about the relative intervals of well-known paging algorithms. These results show that LRU and FIFO are better than FWF, a result expected from practice and experience, yet not fully reflected by the competitive ration model. We also show that the relative interval has another good feature, namely we prove that it reflects the influence of lookahead.

2 Relative Interval Analysis

We introduce a new model for comparing online algorithms. In this model we directly compare two online algorithms, i.e., we do not use the optimal offline algorithm as the baseline of our comparisons. Let $\mathcal{A}$ and $\mathcal{B}$ be two online

Table 1. Summary of the results for relative intervals of several paging algorithms

	LRU	FWF	FIFO	LIFO	LRU-2
LRU		$[-\frac{k-1}{k}, 0]$	$\supseteq [-\frac{k-1}{2k-1}, \frac{k-1}{2k-1}]$	$[-1, \frac{k-1}{k}]$	$\supseteq [-\frac{k-1}{k+1}, \frac{k-1}{2k}]$
FWF	$[0, \frac{k-1}{k}]$		$[0, \frac{k-1}{k}]$		
FIFO	$\supseteq [-\frac{k-1}{2k-1}, \frac{k-1}{2k-1}]$	$[-\frac{k-1}{k}, 0]$		$[-1, \frac{k-1}{k}]$	
LIFO	$[-\frac{k-1}{k}, 1]$		$[-\frac{k-1}{k}, 1]$		
LRU-2	$\supseteq [-\frac{k-1}{2k}, \frac{k-1}{k+1}]$				

algorithms for the same minimization problem, e.g., paging. We define the following two functions:

$$Min_{\mathcal{A},\mathcal{B}}(n) = \min_{|\sigma|=n} \{\mathcal{A}(\sigma) - \mathcal{B}(\sigma)\},$$

and

$$Max_{\mathcal{A},\mathcal{B}}(n) = \max_{|\sigma|=n} \{\mathcal{A}(\sigma) - \mathcal{B}(\sigma)\}.$$

Using these functions we define

$$Min(\mathcal{A},\mathcal{B}) = \liminf_n \frac{Min_{\mathcal{A},\mathcal{B}}(n)}{n}, \quad \text{and} \quad Max(\mathcal{A},\mathcal{B}) = \limsup_n \frac{Max_{\mathcal{A},\mathcal{B}}(n)}{n}.$$

Note that $Min(\mathcal{A},\mathcal{B}) = -Max(\mathcal{B},\mathcal{A})$ and $Max(\mathcal{A},\mathcal{B}) = -Min(\mathcal{B},\mathcal{A})$. Now we are ready to define the *relative interval* of $\mathcal{A}$ and $\mathcal{B}$ as

$$\mathcal{I}(\mathcal{A},\mathcal{B}) = [Min(\mathcal{A},\mathcal{B}), Max(\mathcal{A},\mathcal{B})].$$

This interval gives useful information about the relative performance of $\mathcal{A}$ and $\mathcal{B}$. If $Max(\mathcal{A},\mathcal{B}) > |Min(\mathcal{A},\mathcal{B})|$ then $\mathcal{B}$ has better performance than $\mathcal{A}$ in this model. In this case we say that $\mathcal{B}$ *dominates* $\mathcal{A}$. Also if $Max(\mathcal{A},\mathcal{B})$ is close to 0 then we can conclude that $\mathcal{A}$ is not much worse than $\mathcal{B}$ on any sequences. We can interpret other situations in an analogous way.

We compute the value of $Min(\mathcal{A},\mathcal{B})$ and $Max(\mathcal{A},\mathcal{B})$ for various choices of $\mathcal{A}$ and $\mathcal{B}$. In some cases we obtained bounds or approximation of these values instead. We say that $[\alpha,\beta]$ approximates the relative interval of $\mathcal{A}$ and $\mathcal{B}$ if $Min(\mathcal{A},\mathcal{B}) \leq \alpha$ and $\beta \leq Max(\mathcal{A},\mathcal{B})$. We show this by $\mathcal{I}(\mathcal{A},\mathcal{B}) \supseteq [\alpha,\beta]$.

3 Relative Interval Analysis Applied to Paging Algorithms

In this section we compare some well known paging algorithms using the new model. First we define some other paging algorithms. On a fault, *Last-In-First-Out* (LIFO) evicts the page that is most recently brought to the cache. LIFO does not have a constant competitive ratio [6]. A paging algorithm is called *conservative* if it incurs at most k faults on any sequence that contains at most

k distinct pages. A marking algorithm $\mathcal{A}$ works in phases: all the pages in the cache are unmarked at the beginning of each phase. We mark any page just after the first request to it. When an eviction is necessary, $\mathcal{A}$ should evict an unmarked page. LRU and FIFO are conservative algorithms, while LRU and FWF are marking algorithms. LRU-2 is another paging algorithm proposed by O'Neil et al. for database disk buffering [16]. On a fault, LRU-2 evicts the page whose second to the last request is least recent. If there are pages in the cache that have been requested only once so far, LRU-2 evicts the least recently used among them. Boyar et al. proved that LRU-2 has competitive ratio $2k$ [8].

Lemma 1. *For any two online paging algorithms $\mathcal{A}$ and $\mathcal{B}$,*

$$0 \leq Max(\mathcal{A}, \mathcal{B}) \leq 1.$$

Proof. For any n, there is a sequence σ of length n so that $\mathcal{A}(n) = n$, i.e., $\mathcal{A}$ incurs a fault on every request of σ. This sequence can be obtained by requesting, at each step, the page that is evicted by $\mathcal{A}$ in the previous step. $\mathcal{B}$ incurs at most n faults on every sequence of length n. Therefore $\mathcal{B}(\sigma) \leq n$ and $\mathcal{A}(\sigma) - \mathcal{B}(\sigma) \geq 0$. Thus $\max_{|\sigma|=n}\{\mathcal{A}(\sigma) - \mathcal{B}(\sigma)\} \geq 0$. Since this holds for every n, we have $Max(\mathcal{A}, \mathcal{B}) \geq 0$. For the upper bound, note that for every sequence σ of length n, we have

$$\mathcal{A}(n) \leq n \Rightarrow \mathcal{A}(n) - \mathcal{B}(n) \leq n \Rightarrow \frac{A(n) - B(n)}{n} \leq 1.$$

Therefore $Max(\mathcal{A}, \mathcal{B}) \leq 1$.

Corollary 1. *For any two online paging algorithms $\mathcal{A}$ and $\mathcal{B}$,*

$$-1 \leq Min(\mathcal{A}, \mathcal{B}) \leq 0.$$

Theorem 1. $\mathcal{I}(FWF, LRU) = [0, \frac{k-1}{k}]$.

Proof. Let σ be an arbitrary sequence of length n and consider the partition of σ to phases of marking algorithms. At each phase, FWF incurs exactly k faults while LRU incurs at most k faults. Therefore the cost of LRU on σ is at most the cost of FWF on σ and we have $Min(FWF, LRU) \geq 0$. According to Corollary 1 we have $Min(FWF, LRU) = 0$. At each phase, LRU incurs at least one fault and each phase has length at least k. Therefore $Max(FWF, LRU) \leq \frac{k-1}{k}$. Consider the following sequence for some arbitrary integer m:

$$\sigma = \{p_1 p_2 \cdots p_k p_{k+1} p_2 p_3 \cdots p_k\}^m.$$

We have $FWF(\sigma) = 2k \times m$ and $LRU(\sigma) = k + 1 + 2 \times (m - 1)$ and thus $Max(FWF, LRU) \geq \frac{k-1}{k}$.

Using an analogous argument we can prove the following Theorem.

Theorem 2. $\mathcal{I}(FWF, FIFO) = [0, \frac{k-1}{k}]$.

Lemma 2. *For any any conservative algorithm $\mathcal{A}$ and any online algorithm $\mathcal{B}$, we have $Max(\mathcal{A}, \mathcal{B}) \leq \frac{k-1}{k}$.*

Proof. Let σ be an arbitrary sequence of length n and partition σ into blocks so that $\mathcal{B}$ incurs a fault only on the first request of each block. Therefore each block has at most k distinct pages and $\mathcal{B}$ incurs at most k faults in each block. Let $b_1, b_2, \cdots, b_d$ be the sizes of blocks of σ. Then we have $\mathcal{B}(\sigma) = d$ and $\mathcal{A}(\sigma) \leq \sum_{b_i \leq k} b_i + \sum_{b_i > k} k$. Therefore

$$\frac{\mathcal{A}(\sigma) - \mathcal{B}(\sigma)}{n} \leq \frac{\sum_{b_i \leq k} b_i + \sum_{b_i > k} k - d}{\sum_{b_i \leq k} b_i + \sum_{b_i > k} b_i} \leq \frac{\sum_{b_i \leq k} (b_i - 1) + \sum_{b_i > k} (k-1)}{\sum_{b_i \leq k} b_i + \sum_{b_i > k} k}.$$

If $b_i \leq k$, we have $\frac{b_i - 1}{b_i} \leq \frac{k-1}{k}$ and thus

$$\frac{\sum_{b_i \leq k} (b_i - 1)}{\sum_{b_i \leq k} b_i} \leq \frac{k-1}{k}.$$

Also we have

$$\frac{\sum_{b_i > k} (k-1)}{\sum_{b_i > k} k} \leq \frac{k-1}{k}.$$

Therefore

$$\frac{\mathcal{A}(\sigma) - \mathcal{B}(\sigma)}{n} \leq \frac{k-1}{k}.$$

Since this is true for any σ, we have $Max(\mathcal{A}, \mathcal{B}) \leq \frac{k-1}{k}$.

Theorem 3. $\mathcal{I}(LIFO, LRU) = [-\frac{k-1}{k}, 1]$.

Proof. Since LRU is conservative, according to Lemma 2 we have

$$Max(LRU, LIFO) \leq \frac{k-1}{k} \Rightarrow Min(LIFO, LRU) \geq -\frac{k-1}{k}.$$

Now consider the sequence $\sigma = \{p_1 p_2 \cdots p_k p_{k+1}\}^m$. LRU incurs a fault on every request of σ while LIFO incurs a fault on every kth request. Thus

$$Min(LIFO, LRU) \leq -\frac{k-1}{k} \Rightarrow Min(LIFO, LRU) = -\frac{k-1}{k}.$$

For the other direction consider the sequence $\sigma = p_1 p_2 \cdots p_k p_{k+1} \{p_k p_{k+1}\}^m$. LIFO incurs a fault on every request while LRU only incurs a fault on the first $k+1$ pages. Since m can be arbitrarily large, we have $Max(LIFO, LRU) \geq 1$. According to Lemma 1 we have $Max(LIFO, LRU) = 1$.

The following theorem can be proved in an analogous way.

Theorem 4. $\mathcal{I}(LIFO, FIFO) = [-\frac{k-1}{k}, 1]$.

Lemma 3. $\mathcal{I}(FIFO, LRU) \supseteq [-\frac{k-1}{2k-1}, \frac{k-1}{2k-1}]$.

$$\begin{pmatrix}
p_{k-1} & p_{k-2} & \cdots & p_2 & p_1 & \mathbf{p_{k+1}} & p_1 & p_2 & \cdots & p_{k-1} \\
p_{k-2} & p_{k-3} & \cdots & p_1 & p_{k+1} & \mathbf{p_k} & p_{k+1} & p_1 & \cdots & p_{k-2} \\
p_{k-3} & p_{k-4} & \cdots & p_{k+1} & p_k & \mathbf{p_{k-1}} & p_k & p_{k+1} & \cdots & p_{k-3} \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
p_k & p_{k-1} & \cdots & p_3 & p_2 & \mathbf{p_1} & p_2 & p_3 & \cdots & p_k
\end{pmatrix}$$

Fig. 1. Blocks of sequence σ in the proof of Lemma 3

Proof. $Max(FIFO, LRU)$: Consider the following sequence σ that consists of $k+1$ distinct pages. σ starts with $\sigma_0 = p_1 p_2 \dots p_k$. After this initial subsequence, σ consists of several blocks. Each block starts right after the previous block and contains $2k-1$ requests to k distinct pages. The first k blocks of σ are shown in Fig. 1. The blocks repeat after this, i.e., the $(k+1)$th block is the same as the first block, the $(k+2)$th block is the same as the second block and so on. It is easy to verify that FIFO incurs a fault on the last k requests of each block while LRU only incurs a fault on the middle request of every block. Let σ have m blocks. Then we have $FIFO(\sigma) = k + m \times k$ and $LRU(\sigma) = k + m$. Therefore

$$\frac{FIFO(\sigma) - LRU(\sigma)}{|\sigma|} = \frac{m(k-1)}{k + m(2k-1)},$$

and for sufficiently large values of m, this value becomes arbitrarily close to $\frac{k-1}{2k-1}$.

$Min(FIFO, LRU)$: Consider the following sequence σ' that consists of $k+1$ distinct pages. σ' starts with $\sigma'_0 = p_1 p_2 \dots p_k p_{k-1} p_{k-2} \dots p_1$. After this initial subsequence, σ' consists of m blocks. The first k blocks of σ' are shown in Fig. 2. The blocks repeat after this, e.g., the $(k+1)$th block is the same as the first block. It is easy to verify that LRU incurs a fault on all k requests of each block while FIFO only incurs a fault on the first request of every block. Then we have $LRU(\sigma) = k + m \times k$ and $FIFO(\sigma) = k + m$. Therefore

$$\frac{FIFO(\sigma) - LRU(\sigma)}{|\sigma|} = \frac{-m(k-1)}{k + m(2k-1)},$$

and for sufficiently large values of m, this value becomes arbitrarily close to $-\frac{k-1}{2k-1}$.

Lemma 4. $Max(LRU\text{-}2, LRU) \geq \frac{k-1}{k+1}$.

Proof. Consider the sequence σ obtained by m times repetition of the block $p_1 p_2 \dots p_{k-1} p_k p_k p_{k-1} \dots p_1 p_{k+1} p_{k+1}$. In the first block, LRU incurs $k+1$ faults. In every other block, it only incurs two faults, one on the first request to p_k, and one on the first request to p_{k+1}. Therefore we have $LRU(\sigma) = k+1+2(m-1) = 2m + k - 1$. LRU-2 incurs $k+1$ faults in the first block and $2k$ faults in every other block; it has a hit only on the second requests to p_k and p_{k+1} in each block

$$\begin{pmatrix} \mathbf{p_{k+1}} & p_k & p_{k-1} & \cdots & p_3 & p_2 \\ \mathbf{p_1} & p_{k+1} & p_k & \cdots & p_4 & p_3 \\ \mathbf{p_2} & p_1 & p_{k+1} & \cdots & p_5 & p_4 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \mathbf{p_k} & p_{k-1} & p_{k-2} & \cdots & p_2 & p_1 \end{pmatrix}$$

Fig. 2. Blocks of sequence σ' in the proof of Lemma 3

(other than the first block). Therefore we have LRU-2$(\sigma) = k+1+2k(m-1) = 2km-k+1$. Thus

$$\frac{\text{LRU-2}(\sigma) - LRU(\sigma)}{|\sigma|} = \frac{2km-k+1-2m-k+1}{m(2k+2)} = \frac{m(2k-2)-2k+2}{m(2k+2)},$$

and for sufficiently large values of m, this value becomes arbitrarily close to $\frac{2k-2}{2k+2} = \frac{k-1}{k+1}$.

Lemma 5. *$Max(LRU\text{-}2, LRU) \leq \frac{k-1}{k}$.*

Proof. Let σ be an arbitrary sequence of length n and partition σ to blocks so that LRU incurs a fault only on the first request of each block. Let $B_1, B_2, \ldots, B_d$ be the blocks of σ, and b_i be the size of block B_i. Then we have $LRU(\sigma) = d$ and LRU-2$(\sigma) \leq \sum_{1 \leq i \leq d} b_i$. We show that LRU-2 incurs at most k faults in each block. B_1 contains requests to one page and LRU-2 incurs one fault on it. Consider an arbitrary block B_i for $i > 1$, let p be the first request of B_i, and let $p_1, p_2, \ldots, p_{k-1}$ be the $k-1$ most recently used pages before the block B_i in this order. We have $p \notin P = \{p_1, p_2, \ldots, p_{k-1}\}$, because LRU incurs a fault on p. We claim that each request of B_i is either to p or to a page of P. Assume for the sake of contradiction that B_i contains a request to a page $q \notin \{p\} \cup P$ and consider the first request to q in B_i. All pages $p, p_1, p_2, \ldots, p_{k-1}$ are requested since the previous request to q. Therefore at least k distinct pages are requested since the last request to q and LRU incurs a fault on q. This contradicts the definition of a block. Therefore B_i contains at most k distinct pages.

We claim that LRU-2 incurs at most one fault on every page q in block B_i. Assume that this is not true and LRU-2 incurs two faults on a page q in B_i. Therefore q is evicted after the first request to it in B_i. Assume that this eviction happened on a fault on a request to page r and consider the pages that are in LRU-2's cache just before that request. Since $r \in \{p\} \cup P$ is not in the cache and $|\{p\} \cup P| = k$, there is a page $s \notin \{p\} \cup P$ in the cache. The last request to s is before the last request to p_{k-1} before the block B_i, while the second last request to q is after this request. Therefore LRU-2 does not evict q on this fault, which is a contradiction. Thus, LRU-2 contains

Odd: $p_k\ p_{k-1}\ \ldots\ p_2\ \ p_1\ \ p_{k+1}\ p_k\ p_{k-1}\ \ldots\ \ p_3\ \ p_2$
Even: $p_2\ \ p_3\ \ \ldots\ p_k\ p_{k+1}\ \ p_1\ \ p_2\ \ p_3\ \ \ldots\ p_{k-1}\ p_k$

Fig. 3. A block of sequence σ in the proof of Lemma 6

at most k distinct pages in each block and incurs at most one fault on each page. Hence LRU-2 incurs at most k faults in each block of σ. Therefore

$$\frac{\text{LRU-2}(\sigma) - LRU(\sigma)}{n} \leq \frac{\sum_{b_i \leq k} b_i + \sum_{b_i > k} k - d}{\sum_{b_i \leq k} b_i + \sum_{b_i > k} b_i} \leq \frac{\sum_{b_i \leq k} (b_i - 1) + \sum_{b_i > k} (k-1)}{\sum_{b_i \leq k} b_i + \sum_{b_i > k} k}.$$

If $b_i \leq k$, we have $\frac{b_i - 1}{b_i} \leq \frac{k-1}{k}$ and thus

$$\frac{\sum_{b_i \leq k} (b_i - 1)}{\sum_{b_i \leq k} b_i} \leq \frac{k-1}{k}.$$

Also we have

$$\frac{\sum_{b_i > k} (k-1)}{\sum_{b_i > k} k} \leq \frac{k-1}{k}.$$

Therefore

$$\frac{\text{LRU-2}(\sigma) - LRU(\sigma)}{n} \leq \frac{k-1}{k}.$$

Since this is true for any σ, we have $Max(\text{LRU-2}, LRU) \leq \frac{k-1}{k}$.

Lemma 6. $Min(LRU\text{-}2, LRU) \leq -\frac{k-1}{2k}$.

Proof. Consider the following sequence σ that consists of $k+1$ distinct pages. σ starts with $\sigma_0 = p_1 p_2 \ldots p_k$. After this initial subsequence, σ consists of m blocks. Each block starts right after the previous block. The ith block consists of one of the subsequences shown in Figure 3, depending on the parity of i. It is easy to verify that LRU incurs a fault on the last k requests of each block while LRU-2 only incurs a fault on the middle request of every block, i.e., p_{k+1} in *Odd* blocks and p_1 in *Even* blocks. Then we have $LRU(\sigma) = k + m \times k$ and $\text{LRU-2}(\sigma) = k + m$. Therefore

$$\frac{\text{LRU-2}(\sigma) - LRU(\sigma)}{|\sigma|} = \frac{-m(k-1)}{k + m(2k)},$$

and for sufficiently large values of m, this value becomes arbitrarily close to $-\frac{k-1}{2k}$.

Therefore while Max(LRU-2,LRU) is almost 1, we have proven so far an upper bound of almost -1/2 for Min(LRU-2,LRU). A natural question is whether we can improve this bound, i.e., prove that Min(LRU-2,LRU) is less than -1/2.

We believe that this is not true and prove it for the case that we only have $k+1$ distinct pages (note that all our examples are using $k+1$ distinct pages). While this is a counterintuitive result, in the sense that LRU-2 is preferable in practice it adds to our understanding of the relative advantages of LRU and LRU-2. This results indicates that in the fault rate model LRU is also preferable to LRU-2 and hence additional assumptions need to be made in a model (such as the independency of requests model [15]) that would accurately reflect the superiority of LRU-2 observed in practice.

Lemma 7. *If we have at most $k+1$ distinct pages then Min(LRU-2,LRU)$\geq -1/2$.*

Proof. We call a request a "disparity" if it is a fault for LRU and a hit for LRU-2. Note that only a disparity may reduce the value of Min(LRU-2,LRU). Consider an arbitrary sequence $\sigma = \sigma_1\sigma_2 \ldots \sigma_n$ and an arbitrary page p. Let S be the set of all k distinct pages other than p. We prove that between any two request for p in σ causing a disparity there should be a request to p that is not a disparity. Assume for the sake of contradiction that this is not the case: σ_a and σ_b are disparity requests to p, and there is no request to p between them. Let σ_x be the last request to p before σ_a. Since p_a is a fault for LRU, it has evicted p between p_x and p_a. Therefore all members of S are requested between p_x and p_a. Similarly all pages of S are requested between p_a and p_b. Since p is at LRU-2's cache right before p_a, there should be at least one page in S that is not in its cache at that time. As all pages of S are requested between p_a and p_b, LRU-2 incurs at least one fault in this interval. Let p_y be the last request between p_a and p_b on which LRU-2 incurs a fault. We claim that LRU-2 should evict p on the request p_y. Assume that LRU-2 evicts a page $q \in S$ on the fault p_y. The next request to q would be a fault for LRU-2 and since p_y is its last fault between p_a and p_b and q is requested in this range, we conclude that q has been requested between p_a and p_y. However this means that the second last request to q is after p_x, while the second last request to p is at p_x. Thus LRU-2 should evict p at p_y, and p_b is a fault for LRU-2 which is a contradiction. Hence corresponding to each request that may reduce the value of Min(LRU-2,LRU) there is at least one request that does not. This proves the bound of $-1/2$ for Min(LRU-2,LRU).

3.1 Influence of Lookahead

We demonstrate that the relative interval reflects the effects of lookahead. Let LRU(ℓ) be a modification of LRU defined for a lookahead of size ℓ as follows [1]. On a fault, LRU(ℓ) evicts a page in the cache that is least recently used among the pages that are not in the current lookahead. It is shown [3] that LRU(ℓ) incurs no more faults than LRU on any sequence. Therefore $Min(LRU, LRU(\ell)) = 0$. Now consider the sequence $\sigma = \{p_1 p_2 \ldots p_k p_{k+1}\}^m$. LRU incurs a fault on every request of σ while LRU(ℓ) incurs a fault on every lth request. Hence

$$Max(LRU, LRU(\ell)) \geq 1 - 1/l,$$

and thus LRU(ℓ) dominates LRU.

4 Conclusions and Open Questions

We have introduced a fault rate based metric to compare paging algorithms. Using this metric, we have shown the superiority of LRU and FIFO to FWF. The metric also model reflects the beneficial influence of lookahead.

Several natural open questions remain: filling in the remaining entries in Table 1 as well as refining the bounds that are not tight. Furthermore we believe that the relative interval can be of interest in other online settings and even perhaps for the comparison of offline algorithms.

References

1. Albers, S.: On the influence of lookahead in competitive paging algorithms. Algorithmica 18(3), 283–305 (1997)
2. Albers, S., Favrholdt, L.M., Giel, O.: On paging with locality of reference. Journal of Computer and System Sciences 70 (2005)
3. Angelopoulos, S., Dorrigiv, R., López-Ortiz, A.: On the separation and equivalence of paging strategies. In: Proceedings of the 18th ACM-SIAM Symposium on Discrete Algorithms (SODA 2007), pp. 229–237 (2007)
4. Becchetti, L.: Modeling locality: A probabilistic analysis of LRU and FWF. In: Albers, S., Radzik, T. (eds.) ESA 2004. LNCS, vol. 3221, pp. 98–109. Springer, Heidelberg (2004)
5. Ben-David, S., Borodin, A.: A new measure for the study of on-line algorithms. Algorithmica 11, 73–91 (1994)
6. Borodin, A., El-Yaniv, R.: Online Computation and Competitive Analysis. Cambridge University Press, Cambridge (1998)
7. Borodin, A., Irani, S., Raghavan, P., Schieber, B.: Competitive paging with locality of reference. Journal of Computer and System Sciences 50, 244–258 (1995)
8. Boyar, J., Ehmsen, M.R., Larsen, K.S.: Theoretical evidence for the superiority of LRU-2 over LRU for the paging problem. In: Erlebach, T., Kaklamanis, C. (eds.) WAOA 2006. LNCS, vol. 4368, pp. 95–107. Springer, Heidelberg (2007)
9. Boyar, J., Favrholdt, L.M., Larsen, K.S.: The Relative Worst Order Ratio Applied to Paging. In: ACM-SIAM SODA 2005, pp. 718–727 (2005)
10. Boyar, J., Favrholdt, L.M.: The relative worst order ratio for on-line algorithms. In: Petreschi, R., Persiano, G., Silvestri, R. (eds.) CIAC 2003. LNCS, vol. 2653, Springer, Heidelberg (2003)
11. Boyar, J., Medvedev, P.: The relative worst order ratio applied to seat reservation. In: SWAT: Scandinavian Workshop on Algorithm Theory (2004)
12. Dorrigiv, R., López-Ortiz, A.: A survey of performance measures for on-line algorithms. SIGACT News (ACM Special Interest Group on Automata and Computability Theory) 36(3), 67–81 (2005)
13. Kenyon, C.: Best-fit bin-packing with random order. In: ACM-SIAM SODA 1996, pp. 359–364 (1996)
14. Koutsoupias, E., Papadimitriou, C.: Beyond competitive analysis. SIAM J. Comput. 30, 300–317 (2000)
15. Megiddo, N., Modha, D.S.: ARC: A self-tuning, low overhead replacement cache. In: Proceedings of the 2nd USENIX Conference on File and Storage Technologies (FAST 2003), pp. 115–130 (2003)

16. O'Neil, E.J., O'Neil, P.E., Weikum, G.: The LRU-K page replacement algorithm for database disk buffering. In: Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data, pp. 297–306 (1993)
17. Panagiotou, K., Souza, A.: On adequate performance measures for paging. In: Proceedings of the 38th Annual ACM Symposium on Theory of Computing (STOC 2006), pp. 487–496 (2006)
18. Silberschatz, A., Galvin, P.B., Gagne, G.: Operating System Concepts. John Wiley & Sons, Chichester (2002)
19. Sites, R.L., Agarwal, A.: Multiprocessor cache analysis using ATUM. In: Proceedings of the fifteenth Annual International Symposium on Computer Architecture (ISCA 1988), pp. 186–195 (1988)
20. Sleator, D.D., Tarjan, R.E.: Amortized efficiency of list update and paging rules. Communications of the ACM 28, 202–208 (1985)
21. Youn, N.E.: The k-server dual and loose competitiveness for paging. Algorithmica 11(6), 525–541 (1994)
22. Young, N.E.: Bounding the diffuse adversary. In: Proceedings of the 9th ACM-SIAM Symposium on Discrete Algorithms (SODA 1998), pp. 420–425 (1998)
23. Young, N.E.: On-line paging against adversarially biased random inputs. Journal of Algorithms 37(1), 218–235 (2000)
24. Young, N.E.: On-line file caching. Algorithmica 33(3), 371–383 (2002)

I/O-Efficient Map Overlay and Point Location in Low-Density Subdivisions*

Mark de Berg[1], Herman Haverkort[1], Shripad Thite[2], and Laura Toma[3]

[1] Dept. of Computer Science, Eindhoven University of Technology, the Netherlands
mdberg@win.tue.nl, cs.herman@haverkort.net
[2] Center for the Mathematics of Information, California Institute of Technology, USA
shripad@caltech.edu
[3] Dept. of Computer Science, Bowdoin College, USA
ltoma@bowdoin.edu

Abstract. We present improved and simplified I/O-efficient algorithms for two problems on planar low-density subdivisions, namely map overlay and point location. More precisely, we show how to preprocess a low-density subdivision with n edges in $O(sort(n))$ I/O's into a compressed linear quadtree such that one can:

(i) compute the overlay of two preprocessed subdivisions in $O(scan(n))$ I/O's, where n is the total number of edges in the two subdivisions,
(ii) answer a single point location query in $O(\log_B n)$ I/O's and k batched point location queries in $O(scan(n) + sort(k))$ I/O's.

For the special case where the subdivision is a fat triangulation, we show how to obtain the same bounds with an ordinary (uncompressed) quadtree, and we show how to make the structure fully dynamic using $O(\log_B n)$ I/O's per update. Our algorithms and data structures improve on the previous best known bounds for general subdivisions both in the number of I/O's and storage usage, they are significantly simpler, and several of our algorithms are cache-oblivious.

1 Introduction

The traditional approach to algorithms design considers each atomic operation to take roughly the same amount of time. Unfortunately this simplifying assumption is invalid when the algorithm operates on data stored on disk: reading data from or writing data to disk can be a factor 100,000 or more slower than doing an operation on data that is already present in main memory. Thus, when the data is stored on disk it is usually much more important to minimize the number of disk accesses, rather than to minimize the CPU computation time.

This has led to the study of so-called I/O-efficient algorithms, also known as external-memory or out-of-core algorithms. The by now standard way of analyzing I/O-efficient algorithms is with the model introduced by Aggarwal and Vitter [1]. In this model, a computer has an internal memory of size M and an

* MdB and ST were supported by the Netherlands' Organisation for Scientific Research (NWO).

T. Tokuyama (Ed.): ISAAC 2007, LNCS 4835, pp. 500–511, 2007.

arbitrarily large disk. The data on disk is stored in blocks of size B, and whenever an algorithm wants to work on data not present in internal memory, the block(s) containing the data are read from disk. The I/O-complexity of an algorithm is the number of I/O's it performs—that is, the number of block transfers between the internal memory and the disk. In this model, scanning—reading a set of n consecutive items from disk—can be done in $scan(n) = \lceil n/B \rceil$ I/O's, and sorting takes $sort(n) = \Theta((n/B)\log_{M/B}(n/B))$ I/O's.

One of the main application areas for I/O-efficient algorithms has always been the area of geographic information systems (GIS), because GIS typically work with massive amounts of data and loading all of it into memory is often infeasible. In GIS the data for a particular geographic region is stored as a number of separate thematic layers. There can be a layer storing the road network, a layer storing the river network, a layer storing a subdivision of the region according to land usage or soil type, and so on. To combine the information from two such layers—for example to find the crossings between the river network and the road network—one has to compute the overlay of the layers. Even though the map-overlay problem is one of the most basic algorithmic problems in spatial databases and GIS, which are main application areas for I/O-efficient algorithms, the map-overlay problem has still not been solved satisfactorily in the I/O-model.

Background. In this paper we study the following map overlay problem, also known as the red-blue intersection problem: given a set of non-intersecting blue segments and a set of non-intersecting red segments in the plane, compute all intersections between the red and blue segments. Arge et al. [3] showed how to solve this problem in $O(sort(n) + k/B)$ I/O's, where k is the number of intersections. Even though this is optimal in the worst case, it is not satisfactory for several reasons. First, as pointed out by the authors, their solution is complicated. Hence, the practical value of the solution is unclear. Second, the algorithm presented by Arge et al. [3] uses $\Theta(n \log_{M/B} n/B)$ storage (that is, $\Theta(n/B \cdot \log_{M/B} n/B)$ disk blocks). A randomized solution for computing the intersections in a set of line segments is described by Crauser et al. [10]. They give the same expected I/O-bound of $O(sort(n) + k/B)$ I/O's and linear space under some (realistic) assumptions for M, B, n. We do not know whether this algorithm is practical.

Although the I/O-complexity of the above algorithms is optimal for general sets of line segments, there are important special cases for which this is not clear. For example, the overlay of two triangulations that are suitably stored—in a doubly-connected edge list [14], say—can trivially be computed in $O(n+k)$ time in internal memory, which raises the question whether the overlay of two suitably stored triangulations can be computed in $O(scan(n+k))$ I/O's. In fact, in internal memory one can overlay two subdivisions in $O(n + k)$ time when these subdivisions are connected [15]. This brings us to the topic of our paper: is it possible to do the overlay of two planar maps in $O(scan(n + k))$ I/O's? And can it be done *cache-obliviously* [16], that is, can it be done without specifying the memory size M and the block size B in our algorithms, so that no parameter tuning is necessary and the I/O-behavior is good over all levels in a multilevel memory hierarchy?

Most research into I/O-efficient algorithms has focused on algorithms that are efficient for any worst-case input. However, worst-case inputs are often artificial constructions that do not usually occur in real life. In computational geometry this has led to the study of input models where inputs are assumed to have properties that make them resemble realistic inputs better [13]. The most common assumption in such models is that the objects are *fat*, that is, they are not arbitrarily long and narrow. Fatness has been studied extensively in computational geometry in the recent years, and solutions to many fundamental problems have been improved by exploiting fatness and related notions, see De Berg et al. [13,12] and references therein.

In this paper, we consider two types of subdivisions: fat triangulations and low-density subdivisions. A *δ-fat triangulation* is a triangulation in which every angle is bounded from below by a fixed positive constant δ. A *λ-low-density subdivision* is a subdivision such that any disk D is intersected by at most λ edges whose length is at least the diameter of D, for some fixed constant λ. We believe these two types of subdivisions include most subdivisions encountered in practice, for reasonable values of δ and λ.

The data structures on which we will base our solutions are modifications of the so-called *linear quadtree*, which was introduced by Gargantini [17]. The linear quadtree is a quadtree variant where only the leaf regions are stored, and not the internal nodes. To facilitate a search in the quadtree, a linear order is defined on the leaves based on some space-filling curve; then a B-tree is constructed on the leaves using this ordering—see Section 2 for details. The idea of using linear quadtrees to store planar subdivisions has been used by Hjaltason and Samet [19]. They present algorithms for constructing (or: bulk-loading, as it is often called in GIS) the quadtree, for insertions, and for bulk-insertions. Although their experiments indicate their method performs well in practice, it has several disadvantages. First, the I/O-complexity of their algorithms is analysed in terms of various parameters that depend on the data and the algorithm in a way that is not well-understood. In particular, the performance of their algorithms does not seem to be worst-case optimal. Second, the stopping rule for splitting quadtree cells is based on two user-defined parameters (the maximum depth and a so-called splitting threshold), and so the method is not fully automatic.

Our results. In this paper we show how to overcome these disadvantages for fat triangulations and low-density subdivisions and present improved and simplified algorithms for map overlay and point location in external memory. Our results are based on a quadtree which we define to ensure that (i) each leaf intersects only a constant number of edges of the subdivision, (ii) that we create only $O(n)$ leaves, and (iii) that we can construct the leaves efficiently.

For fat triangulations our quadtree is defined by recursively splitting the unit square into quadrants until all edges that intersect a cell are incident to a common vertex. We prove that this stopping rule yields a quadtree of linear size. Nevertheless, due to the potentially large depth of the quadtree, it is still difficult or impossible to build the quadtree I/O-efficiently by distributing the edges from the root down into the quadtree while splitting nodes as needed. Fortunately our stopping

rule makes a completely different approach possible: we give an algorithm that is simple and elegant—simpler than the algorithm of Hjaltason and Samet [19]—and uses only $O(sort(n))$ I/O's.

For low-density subdivisions we continue splitting until each cell contains only a single bounding-box vertex of any edge. This stopping rule leads to cells with a constant number of edges, but the number of cells cannot be bounded. Therefore we combine the ideas of compressed quadtrees and linear quadtrees to get a *linear compressed quadtree*, rather than a regular quadtree. We show that with the stopping rule just defined, the compressed quadtree has linear size. We also give a construction algorithm that uses only $O(sort(n))$ I/O's.

Once we have proved that these quadtrees have linear size and each leaf region intersects a constant number of edges, our other results come almost for free: overlaying two subdivisions boils down to a simple merge of the ordered lists of quadtree leaves taking $O(scan(n))$ I/O's, point location can be done in $O(\log_B n)$ I/O's by searching in the B-tree built on top of the list of quadtree leaves, and performing k batched point location queries can be done in $O(scan(n) + sort(k))$ I/O's by sorting the points along the space-filling curve and merging the sorted list with the list of quadtree leaves. The results for map overlay apply to pairs of fat triangulations, low-density subdivisions, or low-density sets of line segments, as well as to mixed pairs of maps of these types. The structure for fat triangulations can be made fully dynamic at a cost of $O(\log_B n)$ I/O's per update.

An optimal static structure for point location in general planar subdivisions was already given by Goodrich et al. [18] for the standard I/O-model and by Bender et al. [5] for the cache-oblivious model. Batched point location can be done with $O(sort(n+k))$ I/O's in the I/O-model using the algorithm by Arge et al. [3]. The result on dynamization, however, is new as far as we know: the best known dynamic I/O-efficient point location structures use $O(\log_B^2 n)$ I/O's per query [2] in the I/O-model. All our data structures and query algorithms are cache-oblivious. Our construction and update algorithms for triangulations can be made cache-oblivious, except that updates will then take $O(\log_B n + \frac{1}{B}\log^2 n)$ I/O's. These results constitute the first results for cache-oblivious map overlay, batched point location and dynamic point location.

We omit all proofs from this extended abstract; please refer to the full paper for the proofs.

2 Fat Triangulations

In this section we describe our solution for fat triangulations. A δ-fat triangulation is a triangulation consisting of δ-fat triangles, that is, triangles all of whose angles are at least δ for some fixed constant $\delta > 0$. We assume that $B = \Omega(1/\delta)$ and $M = \Omega(1/\delta^3)$. We assume that all triangulations are triangulations of the unit square $[0,1]^2$. (Our algorithms and proofs extend to triangulations of convex regions—we leave the details for the full paper.) We can show the following:

Theorem 1. *Let $\mathcal{F}$ be a δ-fat triangulation with n edges. Knowing the memory size M and the block size B, we can construct, in $O(sort(n/\delta^2))$ I/O's, a linear*

quadtree for $\mathcal{F}$ that stores $O(n/\delta^2)$ cell-edge intersections. With this structure we can perform the following operations:

(i) **Map overlay:** *Given two δ-fat triangulations with n triangles in total, each stored in such a linear quadtree, we can find all pairs of intersecting triangles in $O(scan(n/\delta^2))$ I/O's.*
(ii) **(Batched) point location:** *for any query point p we can find the triangle of $\mathcal{F}$ that contains p in $O(\log_B(n/\delta))$ I/O's, and for any set P of k query points we can find for each point $p \in P$ the face of $\mathcal{F}$ that contains p in $O(scan(n/\delta^2) + sort(k))$ I/O's.*
(iii) **Updates:** *Inserting a vertex, moving a vertex, deleting a vertex, and flipping an edge can all be done in $O((\log_B n)/\delta^4)$ I/O's.*

In the cache-oblivious model the same bounds hold, except that updates then take $O((\log_B n + \frac{1}{B}\log^2 n)/\delta^4)$ I/O's.

2.1 The Quadtree Subdivision for Fat Triangulations

A quadtree is a hierarchical subdivision of the unit square into quadrants, where the subdivision is defined by a criterion to decide what quadrants are subdivided further, and what quadrants are leaves of the hierarchy. A *canonical square* is any square that can be obtained by recursively splitting the unit square into quadrants. For a canonical square σ, let $mom(\sigma)$ denote its parent, that is, the canonical square that contains σ and has twice its width. The leaves of the quadtree form the quadtree subdivision; that is, a *quadtree subdivision* for a set of objects in the unit square is a subdivision into disjoint canonical squares (*quadtree cells*), such that each cell obeys the stopping rule while its parent does not. The stopping rule we use is as follows:

> *Stopping rule for fat triangulations:* Stop splitting when all edges intersecting the cell σ under consideration are incident to a common vertex.

Note that the stopping rule includes the case were σ is not intersected by any edges. We can prove the following:

Lemma 1. *Let $\mathcal{F}$ be a δ-fat triangulation of the unit square with n edges. Then the stopping rule defined above leads to a quadtree subdivision with $O(n/\delta)$ cells, each intersected by at most $2\pi/\delta$ triangles.*

2.2 Storing the Quadtree Subdivision and the Triangulation

We will store the quadtree subdivision defined above as a so-called *linear quadtree* [17]. To this end, we define an ordering on the leaf cells of the quadtree subdivision. The ordering is based on a space-filling curve defined recursively by the order in which it visits the quadrants of a canonical square. We will use the Z-*order space-filling curve* for this, which visits the quadrants in the order bottom left, top left,

bottom right, top right, and within each quadrant, the Z-order curve visits its subquadrants recursively in the same order. Since the intersection of every canonical square with this curve is a contiguous section of the curve, this yields a well-defined ordering of the leaf cells of the quadtree subdivision. We call the resulting order the Z-order.

The Z-order curve not only orders the leaf cells of the quadtree subdivision, but it also provides an ordering for any two points in the unit square—namely the Z-order of any two disjoint canonical squares containing the points. (We assume that canonical squares are closed at the bottom and the left side, and open at the top and the right side.) The Z-order of two points can be determined as follows. For a point $p = (p_x, p_y)$ in $[0,1\rangle^2$, define its Z-index $Z(p)$ to be the value in the range $[0,1\rangle$ obtained by interleaving the bits of the fractional parts of p_x and p_y, starting with the first bit of p_x. The value $Z(p)$ is sometimes called the *Morton block index* of p. The Z-order of two points is now the same as the order of their Z-indices [19]. The Z-indices of all points in a canonical square σ form a subinterval $[z_1, z_2\rangle$ of $[0,1\rangle$, where z_1 is the Z-index of the bottom left corner of σ. Note that any subdivision of the unit square $[0,1\rangle^2$ into k leaf cells of a quadtree corresponds directly to a subdivision of the unit interval $[0,1\rangle$ of Z-indices into k subintervals.

A simple (but novel) way of storing a triangulation in a linear quadtree is now obtained by storing all cell-triangle intersections in a B-tree [9]: each cell-triangle intersection $(\sigma, \triangle)$ of a cell σ corresponding to the Z-index interval $[z_1, z_2\rangle$ is represented by storing triangle $\triangle$ with key z_1. With this way of storing the linear quadtree, the leaf cells of the quadtree are stored implicitly: each pair of consecutive different keys z_1 and z_2 constitutes the Z-index interval of a quadtree leaf cell. For a cache-oblivious solution we can use a cache-oblivious B-tree [5,6,7].

In the remainder we will sometimes need to compute or compare Z-indices. Whether this takes constant time depends on the operations allowed by the model of computation. In any case, since we care mainly about I/O-efficiency, such computations do not effect the analysis of our algorithms.

2.3 Building the Quadtree I/O-Efficiently

The natural algorithm to build a quadtree would take a set of triangles and a canonical square (initially all triangles and the unit square) as input, check if the condition of the stopping rule is satisfied, and if not, distribute the triangles among the four children and subdivide the children recursively. Unfortunately, this algorithm takes $O(n^2)$ time and $O(n^2/B)$ I/O's, as the quadtree may have height $O(n)$. Below we describe a faster algorithm that computes the leaf cells that result with our stopping rule directly, using local computations instead of a top-down approach.

For any vertex v of the given triangulation $\mathcal{F}$, let $star(v)$ be the *star* of v in $\mathcal{F}$; namely, it is the set of triangles of $\mathcal{F}$ that have v as a vertex. Recall that a canonical square is any square that can be obtained by recursively subdividing the unit square into quadrants. For a set S of triangles inside the unit square, we say that a canonical square of σ is *active* in S if it lies completely inside S and all edges from S that intersect σ are incident to a common vertex, while $mom(\sigma)$

intersects multiple edges of S that are not all incident to a common vertex. Thus the cells of the quadtree subdivision we wish to compute for $\mathcal{F}$ are exactly the active canonical squares in $\mathcal{F}$. We can prove the following:

Lemma 2. *Let $\triangle = (u, v, w)$ be a triangle of $\mathcal{F}$ and σ a canonical square that intersects $\triangle$. Then σ is active in $\mathcal{F}$ if and only if σ is active in $star(u)$, $star(v)$ or $star(w)$.*

On the basis of this lemma we can construct the linear quadtree as follows:

1. Compute an adjacency list for each vertex.
2. Scan the adjacency lists for all vertices: for each vertex u load its adjacency list in memory and compute the active cells of $star(u)$, with for each cell σ the triangles that intersect σ. Output each triangle with the key z_1 of the Z-index interval $[z_1, z_2\rangle$ that corresponds to σ.
3. Sort the triangles by key, removing duplicates.
4. Build a (cache-oblivious) B-tree on the list of triangles with their keys.

Lemma 3. *The quadtree described above for a δ-fat triangulation with n edges can be constructed with $O(sort(n/\delta^2))$ I/O's.*

Eliminating superfluous cells. The I/O-complexity of the construction and the storage requirements in practice can be reduced with an easy optimization: we merge all active cells that lie properly inside triangles with their successors or predecessors in the Z-order. In fact, in step 2 of the algorithm, we will not even output such cells. Instead we only output triangle-key pairs for triangle-cell intersections such that an edge of the triangle intersects the cell. We sort these triangle-key pairs, and then scan them. Whenever two consecutive triangles have different keys z_1 and z_2, we identify the most significant bit that differs between them. Let z be the lowest Z-index in $[z_1, z_2]$ for which this bit has value 1. For each triangle stored with key z_2, we now replace its key by z. Thus all cells with Z-indices in $[z_1, z\rangle$ are merged with each other, and all cells with z-values in $[z, z_2]$ are merged with each other.

Since the interval $[z_1, z\rangle$ of the Z-order curve covers a connected area in the plane, all cells in the range $[z_1, z\rangle$ that do not intersect any edge must be completely contained in a triangle that already intersects the cell that starts with Z-index z_1. Similarly, $[z, z_2]$ and the cell that starts with z_2 together cover a connected area in the plane, so the triangles that intersect it must have been stored with key z_2 already. Hence no more triangles need to be stored as a result of merging cells.

Updates. We support the following operations: inserting a vertex, moving a vertex, deleting a vertex, and flipping an edge. By Lemma 2, all leaf cells that intersect a triangle $\triangle = (u, v, w)$ can be computed from the local quadtrees of $star(u)$, $star(v)$ and $star(w)$. Since the size of $star(u)$, $star(v)$ and $star(w)$ is $O(1/\delta)$, by Lemma 1 the total number of cells that intersect $\triangle$ is $O(1/\delta^3)$. Since each of the supported operations changes only $O(1/\delta)$ triangles, we can compute the structure of the quadtree locally in the area of the update, and determine what the

changes entail for the data stored on the disk. All changes can thus be made in $O((\log_B n)/\delta^4)$ I/O's when a normal B-tree is used, and in $O((\log_B n + \frac{1}{B}\log^2 n)/\delta^4)$ I/O's when a cache-oblivious B-tree is used [5,6,7].

2.4 Overlaying Maps and Point Location

Lemma 4. *The linear quadtree for δ-fat triangulations as described above supports map overlay in $O(scan(n/\delta^2))$ I/O's, and point location in $O(\log_B(n/\delta))$ I/O's, where n is the number of points in the triangulation. Batched point location for k points takes $O(scan(n/\delta^2) + sort(k))$ I/O's.*

Proof. Each triangulation's quadtree, or rather, subdivision of the Z-order curve, is stored on disk as a sorted list of Z-indices with triangles. To overlay the two triangulations, we will scan the two quadtrees simultaneously in Z-order, at any point keeping in memory the triangles stored with the last key read from the first list and those stored with the last key read from the second list. Starting from the beginning of the lists, we repeat the following until both lists have been read completely: we read the next key from the list with the smallest unread key, we load all triangles stored with that key into memory, and we compute the intersections with the triangles in memory that were read from the other list. The input has size $O(n/\delta^2)$. The output consists of $O(n/\delta)$ intersections since a δ-fat triangulation has density $O(1/\delta)$ [13], which implies the claim.

To perform point location with a point p, we compute the Z-index $Z(p)$ of p and search the B-tree for the triangles with the highest keys less than or equal to $Z(p)$. To do batched point location, we sort the set P of query points by Z-index, and scan the leaves of the B-tree and P in parallel (similar to the overlay operation as described above). □

3 Low-Density Subdivisions

In this section we describe our solution for storing planar low-density subdivisions. For a planar object o, let $\mathrm{diam}(o)$ denote the diameter of the smallest enclosing disk of o. The *density* of a set S of objects in the plane is the smallest number λ such that the following holds: any disk D is intersected by at most λ objects $o \in S$ such that $\mathrm{diam}(o) \geq \mathrm{diam}(D)$ [13]. We say that a planar subdivision $\mathcal{F}$ has density λ if its edge set has density λ. In other words, any disk D is intersected by at most λ edges whose length is at least the diameter of D. We assume that $B = \Omega(\lambda)$, and that the input lies in the unit square $[0,1]^2$. In this section we describe the following result.

Theorem 2. *Let $\mathcal{F}$ be a subdivision or a set of non-intersecting line segments of density λ with n edges. Knowing the memory size M and the block size B, we can construct, in $O(sort(\lambda n))$ I/O's, a linear compressed quadtree for $\mathcal{F}$ with $O(n)$ cells that each intersect $O(\lambda)$ edges. With this structure we can perform the following operations:*

(i) **Map overlay:** *If we have two subdivisions (or sets of non-intersecting line segments) of density λ with n edges in total, both stored in such a linear compressed quadtree, then we can find all pairs of intersecting edges in $O(scan(\lambda n))$* I/O*'s.*

(ii) **(Batched) point location:** *for any query point p we can find the face of $\mathcal{F}$ that contains p in $O(\log_B n)$* I/O*'s, and for any set P of k query points we can find for each point $p \in P$ the face of $\mathcal{F}$ that contains p in $O(scan(\lambda n) + sort(k))$* I/O*'s.*

The data structure, the overlay algorithm and the query algorithms are cache-oblivious. (The algorithm that constructs *the data structure is not.)*

Any set of disjoint δ-fat triangles in the plane has density $O(1/\delta)$ [13]. Thus the results of this section can be used for δ-fat triangulations. However, the solution from the previous section is simpler and dynamic.

Below we explain our data structure, and how to construct it. The query algorithms are the same as described in the previous section.

3.1 The Compressed Quadtree Subdivision for Low-Density Maps

Let $\mathcal{F}$ be a subdivision of the unit square with n edges and of density λ. In general it is impossible to construct a standard quadtree on $\mathcal{F}$ consisting of a linear number of cells that are each intersected by a constant number of edges. Indeed, in a general subdivision of the unit square there can be many vertices arbitrarily close together, even if the subdivision has constant density. To overcome this problem we shall use a so-called *compressed quadtree*.

Let $\mathcal{G}$ be the set of vertices of the axis-parallel bounding boxes of the edges of $\mathcal{F}$. This set has a nice property:

Lemma 5 ([11]). *Any square σ that does not contain any bounding-box vertex of an object in a set S with density λ, intersects $O(\lambda)$ objects from S.*

We now construct a quadtree for $\mathcal{F}$ with the following stopping rule.

> *Stopping rule for low-density subdivisions:* Stop splitting when the cell σ under consideration contains at most one point from $\mathcal{G}$.

Consider the quadtree that we get from this stopping rule. Its cells intersect $O(\lambda)$ edges, but the number of cells cannot be bounded. Hence, we compress the quadtree [20], by building it as follows. We recursively subdivide each canonical square σ that contains more than one point from $\mathcal{G}$ into *five* regions. Let σ' be the smallest canonical square that contains all points of $\sigma \cap \mathcal{G}$. The first region is the *donut* $\sigma \setminus \sigma'$. The remaining four regions are the four quadrants of σ'. Note that the first region does not contain any points of $\mathcal{G}$, so it is never subdivided further. When $\sigma' = \sigma$, the first region is skipped; when σ' is smaller than σ, we call $\sigma \setminus \sigma'$ a *proper donut*.

Lemma 6. *Let $\mathcal{F}$ be a subdivision of the unit square with n edges and of density λ. Then a compressed quadtree subdivision based on the stopping rule defined above has $O(n)$ cells, and each cell is intersected by at most $O(\lambda)$ edges.*

3.2 Storing the Compressed Quadtree Subdivision and the Low-Density Map

We store the cell-edge intersections of the compressed quadtree subdivision in a list sorted by the Z-order of the cells, indexed by a (cache-oblivious) B-tree. The only difference with the previous section is that we now have to deal with donuts as well as square cells. Recall that a canonical square (a square that can be obtained from the unit square by a recursive partitioning into quadrants) corresponds to an interval on the Z-order curve. For a donut this is not true. However, a donut corresponds to at most two such intervals, because a donut is the set-theoretic difference of two canonical squares. Thus the solution of the previous section can be applied if we represent each donut by two intervals $[z_1, z_2\rangle$ and $[z_3, z_4\rangle$; edges intersecting the first part of the donut are stored with key z_1 and edges intersecting the second part are stored with key z_3. As described in Section 2.3, we merge cells that do not intersect any edge with their immediate successors or predecessors in the Z-order. We call the resulting structure—the B-tree on the cell-edge intersections whose keys imply a compressed quadtree subdivision—a *linear compressed quadtree*. Map overlay and point location are done in exactly the same way as with the linear quadtree described earlier.

3.3 Building the Compressed Quadtree I/O-Efficiently

We construct the leaves of the compressed quadtree, or rather, the corresponding subdivision of the Z-order curve, as follows. We sort $\mathcal{G}$ into Z-order, and scan the sorted points. For each pair of consecutive points, say u and v, we construct their lowest common ancestor $lca(u, v)$ by examining the longest common prefix of the bit strings representing $z(u)$ and $z(v)$. We output the five Z-indices that bound and separate the Z-order intervals of the four children of $lca(u, v)$. To complete the subdivision of the Z-order curve, we sort the output into a list $\mathcal{L}$ by Z-order, removing duplicates.

Lemma 7. *The above algorithm generates a subdivision of the* Z*-order curve that corresponds to a compressed quadtree on $\mathcal{G}$ in $O(sort(n))$* I/O*'s.*

Having constructed the compressed quadtree subdivision, we now distribute the edges in $\mathcal{F}$ to the faces of the quadtree subdivision, or rather, to the corresponding sections of the Z-order curve.

To do so, we first build a B-tree on the subdivision of the Z-order curve as computed above. We then load the (roughly) M/B nodes of the B-tree, that reside $\log_B(M/B)$ levels below the root, in memory. Note that each of these nodes covers a certain section of the Z-order curve, and together they form a subdivision of the Z-index interval covered by the root. We assign an output stream to each of these nodes, and reserve a buffer of one block in memory for each of them. We now read the edges from the input one by one, and distribute each edge to the output streams of the nodes whose section of the Z-order curve is intersected by the edge. Note that each edge may be copied to several streams. Once all edges have been

read, we distribute the edges in each node's stream recursively into the subtree rooted at that node.

After distributing all edges recursively to the leaves, we collect all edge-cell intersections, ordered by the z-indices of the cells, and put a new b-tree on top of them. Each cell σ without any intersecting edges is merged with the cells that precede or follow it in the z-order, up to a cell that stores an edge of the face of $\mathcal{F}$ that contains σ (see Section 2.3 for an explanation).

Lemma 8. *The compressed quadtree as defined above for a subdivision of density λ with n edges can be constructed with $O(sort(\lambda n))$ i/o's.*

4 Conclusions

We described how one can efficiently store and overlay planar maps in the i/o-model of computation. Our algorithms work for planar maps that are fat triangulations or have low density. The solution for triangulations is based on quadtrees, is considerably simpler than previous solutions, and supports even dynamic maintenance of the planar maps under updates. The second construction, for low density planar maps, is based on compressed quadtrees and is somewhat more complicated; however our analysis gives a better dependency on the parameter that describes the input. Unfortunately it is not clear if the construction algorithm can be made cache-oblivious and if the structure can be made to support updates.

Both constructions use linear space, improving on the previous space bound of $O(n \log_{M/B} n)$ of Arge et al. [3]. Which of our two structures would give the most compact data structure for triangulations in practice remains to be seen. The first approach's dependency on the fatness may be better than it seems (perhaps an analysis in terms of *average* fatness is possible), while the second approach may introduce many guards (a triangulation of n vertices has roughly $3n$ edges and thus roughly $6n$ extra bounding box vertices as guards).

Our data structures can be used for range searching queries. In general this would not be very efficient, but we believe it is possible to achieve good bounds for *approximate range searching* [4]. However, the data structure for low-density subdivisions as presented in this paper does not give good bounds immediately, and needs to be subjected to some post-processing for this purpose. The post-processing also reduces the size of the data structure to $O(n)$, independent of the density λ. We are currently working out the details.

Acknowledgement. We thank Sariel Har-Peled for his extensive contribution.

References

1. Aggarwal, A., Vitter, J.S.: The input/output complexity of sorting and related problems. Commun. ACM 31, 1116–1127 (1988)
2. Arge, L., Vahrenhold, J.: I/O-efficient dynamic planar point location. In: Proc. 16th Annu. ACM Symp. Comput. Geom., pp. 191–200 (2000)

3. Arge, L., Vengroff, D.E., Vitter, J.S.: External-memory algorithms for processing line segments in geographic information systems. In: Proc. 3th Annu. European Symp. Algorithms, pp. 295–310 (1995)
4. Arya, S., Mount, D.M.: Approximate range searching. Comput. Geom. Theory Appl. 17, 135–152 (2000)
5. Bender, M., Cole, R., Raman, R.: Exponential structures for efficient cache-oblivious algorithms. In: Widmayer, P., Triguero, F., Morales, R., Hennessy, M., Eidenbenz, S., Conejo, R. (eds.) ICALP 2002. LNCS, vol. 2380, pp. 195–207. Springer, Heidelberg (2002)
6. Bender, M.A., Demaine, E.D., Farach-Colton, M.: Cache-oblivious B-trees. In: Proc. 41th Annu. IEEE Symp. Found. Comput. Sci., pp. 339–409 (2000)
7. Brodal, G.S., Fagerberg, R., Jacob, R.: Cache oblivious search trees via binary trees of small height. In: Proc. 13th ACM-SIAM Symp. Discrete Algorithms, pp. 39–48 (2002)
8. Chiang, Y.-J., Goodrich, M.T., Grove, E.F., Tamassia, R., Vengroff, D.E., Vitter, J.S.: External-Memory Graph Algorithms. In: Proc. 6th ACM-SIAM Symp. Discrete Algorithms, pp. 139–149 (1995)
9. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms. MIT Press / McGraw-Hill, Cambridge, Mass (2001)
10. Crauser, A., Ferragina, P., Mehlhorn, K., Meyer, U., Ramos, E.: Randomized external-memory algorithms for some geometric problems. Comput. Geom. Theory Appl. 11(3), 305–337 (2001)
11. de Berg, M.: Linear size binary space partitions for uncluttered scenes. Algorithmica 28, 353–366 (2000)
12. de Berg, M.: Improved bounds on the union complexity of fat objects. In: Proc. 25th Conf. Found. Soft. Tech. Theoret. Comput. Sci., pp. 116–127 (2005)
13. de Berg, M., Katz, M.J., Stappen, A.v., Vleugels, J.: Realistic input models for geometric algorithms. Algorithmica 34, 81–97 (2002)
14. de Berg, M., van Kreveld, M., Overmars, M.H., Schwarzkopf, O.: Computational Geometry: Algorithms and Applications, 2nd edn. Springer, Heidelberg (2000)
15. Finke, U., Hinrichs, K.: Overlaying simply connected planar subdivisions in linear time. In: Proc. 11th Annu. ACM Symp. Comput. Geom., pp. 119–126 (1995)
16. Frigo, M., Leiserson, C.E., Prokop, H., Ramachandran, S.: Cache-oblivious algorithms. In: Proc. 40th Annu. IEEE Symp. Found. Comput. Sci., pp. 285–298 (1999)
17. Gargantini, I.: An effective way to represent quadtrees. Commun. ACM 25(12), 905–910 (1982)
18. Goodrich, M.T., Tsay, J.-J., Vengroff, D.E., Vitter, J.S.: External-memory computational geometry. In: Proc. 34th Annu. IEEE Symp. Found. Comput. Sci., pp. 714–723 (1993)
19. Hjaltason, G.R., Samet, H.: Speeding up construction of PMR quadtree-based spatial indexes. VLDB Journal 11, 137–190 (2002)
20. Samet, H.: Spatial Data Structures: Quadtrees, Octrees, and Other Hierarchical Methods. Addison-Wesley, Reading, MA (1989)

Geometric Streaming Algorithms with a Sorting Primitive

Eric Y. Chen

School of Computer Science, University of Waterloo, Waterloo, ON, Canada, N2L 3G1
y28chen@cs.uwaterloo.ca

Abstract. We solve several fundamental geometric problems under a new streaming model recently proposed by Ruhl et al. [2,12]. In this model, in one pass the input stream can be scanned to generate an output stream or be sorted based on a user-defined comparator; all intermediate streams must be of size $O(n)$. We obtain the following geometric results for any fixed constant $\epsilon > 0$:

- We can construct 2D convex hulls in $O(1)$ passes with $O(n^\epsilon)$ extra space.
- We can construct 3D convex hulls in $O(1)$ expected number of passes with $O(n^\epsilon)$ extra space.
- We can construct a triangulation of a simple polygon in $O(1)$ expected number of passes with $O(n^\epsilon)$ extra space, where n is the number of vertices on the polygon.
- We can report all k intersections of a set of 2D line segments in $O(1)$ passes with $O(n^\epsilon)$ extra space, if an intermediate stream of size $O(n + k)$ is allowed.

We also consider a weaker model, where we do not have the sorting primitive but are allowed to choose a scan direction for every scan pass. Here we can construct a 2D convex hull from an x-ordered point set in $O(1)$ passes with $O(n^\epsilon)$ extra space.

1 Introduction

Nowadays, applications with massive data sets are emerging rapidly in different areas, such as internet applications, geographic information systems, and sensor networks. Researchers have thus proposed algorithms that use small amounts of memory space under different space-conscious models. In in-place algorithms (e.g. see [4]), all input data are stored in memory, and the extra amount of memory used by the program is small or even constant. However, such algorithms are not suitable for data sets, which are larger than the size of memory. These massive data sets are considered under one-pass streaming models and multi-pass streaming models. In the one-pass streaming model, all input data are accessed once sequentially by the program, and the amount of memory used must be sublinear in the size of input or even constant. Algorithms under this model can process data sets larger than the size of the memory, but most of them

T. Tokuyama (Ed.): ISAAC 2007, LNCS 4835, pp. 512–524, 2007.

can only compute approximate solutions. An alternative model is the multi-pass streaming model. In this model, the input data can be accessed sequentially multiple times. Algorithms under this model can compute exact solutions, but typically the number of passes taken by these algorithms grows when the size of input gets larger. In computational geometry, fundamental problems have been considered in all these models [4,5,6]. Not only efficient algorithms are proposed, but lower bounds [6] are also proved for several problems.

Sorting is one of the most studied general-purpose problems for massive data sets. Consequently, sorting is a fully optimized primitive under most applications and operating systems. Ruhl et al. [2,12] recently proposed a practical streaming model augmented with a sorting primitive, which will be defined precisely at the end of this section. Roughly speaking, unlike the previous multi-pass streaming models, we can work with intermediate streams and can sort a stream in a single round; we want to minimize the number of rounds.

Note that, as Ruhl et al. [2,12] showed, many parallel circuits, and consequently many parallel algorithms, can be simulated in the streaming model augmented with a sorting primitive. Several known geometric algorithms [1] can be simulated directly. However, all of the transformed algorithms take $O(\text{polylog}\, n)$ passes. Our algorithms are the first solutions only taking $O(1)$ passes. [1]

In this paper, we study several of the most fundamental problems in computational geometry [8,10,11] in the streaming model augmented with a sorting primitive. For any fixed ϵ, all of the following problems are solved with a constant number of passes and $O(n^\epsilon)$ space. These are the first results for these problems that achieve simultaneously a small number of passes and a reasonably small amount of space in a realistic streaming model. Specifically, these problems are: 1. constructing the convex hull from a set of 2D points, 2. constructing the convex hull from a set of 3D points, 3. constructing a triangulation of a simple polygon. Some of these problems have been even proved unsolvable with a constant number of passes in the original multi-pass streaming model [6]. If the size of the intermediate data stream is allowed to be larger, namely $O(n + k)$, we can solve a fourth problem: report all k intersections of a set of 2D line segments with a constant number of passes and $O(n^\epsilon)$ space in memory.

In a weaker model, which will also be defined precisely at the end of this section, we can still construct a 2D convex hull from an x-ordered point set in a constant number of passes with $O(n^\epsilon)$ space in memory. This has also been proved unsolvable in the multi-pass streaming model.

Some of the techniques we use are standard. For example, for the 3D convex hull and segment intersection problems, we adapt well-known random sampling techniques. To solve the triangulation problem, however, we need to introduce some new geometric observations and use a combination of several ideas.

[1] Indeed, we have just learned that Lammersen and Sohler [9] have independently considered geometric algorithms in the same stream-sort model, but they were only able to obtain $O(\text{polylog}\, n)$ algorithms for the 2D convex hull problem.

1.1 The Streaming Model with a Sorting Primitive

In this subsection, we precisely define the streaming model with a sorting primitive (*stream-sort* model for short), describe one divide-and-conquer technique that we will use throughout in this paper, and also define a weaker model, which we call the *direction-flexible* streaming model.

In the stream-sort model, the input data are given in one data stream. There are two ways to access these data. One is the scan pass. The input stream is scanned sequentially, and one output stream is generated. The other is the sorting pass. The input stream is sorted based on a user-defined comparator, and data are sorted in the output stream based on that order. The generated output stream of data is called an *intermediate* stream. In the next scan, this intermediate stream becomes the input stream, and another output stream is generated. All intermediate streams must be of size $O(n)$. At the end, we count how many passes are used in total, and how much space is used in memory by the program in the scan passes.

Divide-and-conquer is a general strategy commonly used under this model. With this strategy, the data set of the problem is divided into data sets for several subproblems. We describe a technique that can help us solve all subproblems simultaneously in one pass.

This technique can be described as follows. Given a stream containing multiple independent data sets and one data set per subproblem, if the elements of each data set are grouped together in the stream, we can process these data sets one after another in a single pass. In the scan pass, after scanning over the data set for one subproblem, we reset memory for the data set of the next subproblem. Using a sorting pass, we can group the element for the same data set together. This can be done by adding a field in each element to identify which data set it belongs to. Storing this extra field only lengthens the stream by $O(n)$.

Now we treat the recursive calls in a divide-and-conquer algorithm as a tree structure. Each node represents one subproblem we need to solve. We can solve all problems in the same level of the recursion tree in the same pass. Therefore, we can bound the number of passes by the number of levels in the recursion tree.

Since the sorting pass is the most expensive part, we define a simpler and self-contained model in which the sorting pass is not allowed. Instead, we can only choose a direction to scan the stream (forward or backward). We call this model the *direction-flexible* model. This model is weaker than the stream-sort model. For example, it is impossible to sort data in order in a constant number of passes in the direction-flexible model, if the extra space allowed is sublinear.

2 Convex Hulls

2D convex hulls. As the first example, we sketch a simple solution to the problem of constructing the convex hull of 2D points, which is equivalent to constructing the lower envelope of a set of halfplanes in dual space [8]. We divide the halfplanes into B groups and compute the lower envelope of each group

recursively. The B lower envelopes can be merged in a constant number of passes in the stream-sort model by a sweep from left to right. By setting $B = O(n^\epsilon)$, we finish the recursion in $O(1)$ levels. With the technique in Section 1.1, we can build the hull in a constant number of rounds.

3D convex hulls. Next, we consider the 3D convex hull problem and highlight another useful technique: random sampling.

Given a set of 3D points, we transform them into dual space. Each point in primal space maps to a halfspace in dual space. Constructing the convex hull of a set of 3D points is equivalent to constructing the lower envelope of a set of 3D halfspaces [8] in dual space. We first describe our algorithm in the traditional memory model, and then modify it to fit in the stream-sort model. The notations are defined as follows. The cell Δ_f defined by a triangle f refers to the vertical prism underneath f. The set of planes intersecting Δ_f is denoted by H_f. In the algorithm, the input H is a set of n 3D halfspaces, and the output E is the set of faces in the lower envelope. (See [10] for the definition of the *canonical triangulation.*)

Algorithm. 3D_Envelope(H)
Initialize an empty set R
If $|H| \leq B$
    Solve the problem directly in memory and return the answer
Repeat
    Sample a random subset R of size B in H
    Find the lower envelope E_R of R
    Build the canonical triangulation T for E_R
Until $\sum_{f \in T} |H_f| = O(n)$ and $\max_{f \in T} |H_f| = O((n/B) \log B)$
For each $f \in T$
    E_f = 3D_Envelope(H_f)
Merge all the E_f's to form E and return E

The set R is a randomly selected subset. By the analysis from Clarkson and Shor [7] for randomly selected samples, it is known that the expected value of $\sum_{f \in T} |H_f|$ is $O(n)$. Therefore, we have $\sum_{f \in T} |H_f| \leq cn$ with probability greater than a constant, for a sufficiently large constant c. It is also known [10] that $\max_{f \in T} |H_f| \leq c'(n/B) \log B$ with probability greater than a constant, for a sufficiently large constant c'. Therefore, the expected number of iterations before both conditions satisfied is constant.

We modify this algorithm to fit in the stream-sort model. We set B to n^ϵ. Thus, R and E_R takes $O(n^\epsilon)$ space in memory. The operation for choosing the set R can be done in one scan pass. Constructing E_R can be done in memory using $O(n^\epsilon)$ extra space.

By keeping T in memory, verifying the conditions to terminate the loop can be done by one scan pass. One iteration of the first loop can be done in $O(1)$ passes with $O(n^\epsilon)$ extra space.

For the second loop structure, we proceed as follows. We create a copy of h for each cell Δ_f intersected by the halfspace h. We also attach a label to each copy to identify the corresponding cell. This operation can be done in one scan pass. In the following sorting pass, we use the attached label as the key. All halfspaces are grouped together in the data stream. With the technique introduced in Section 1.1, all subproblems in the same level of the recursion tree are solved simultaneously in one scan passes.

Because the total number of intersections between halfspaces and cells is $O(n)$, the duplication of halfspaces only lengthens the size of the intermediate stream by a constant factor times. The size of any subproblem is $O(n^{1-\epsilon} \log n)$, since $B = n^{\epsilon}$. The number of levels of the recursion tree is constant, since the size of the intermediate stream increases by a constant factor every round. Therefore, all intermediate streams are of size $O(n)$.

Because the first loop in the above algorithm terminates in a constant expected number of iterations, this algorithm takes $O(1)$ expected number of passes in one round. Therefore, the expected total number of passes is also $O(1)$.

The merging step at the end of the algorithm can be done by a sorting pass and a scan pass. In the sorting pass, we use the planes containing facets as the key and group all facets in the same plane together.

Theorem 1. *Given a set of 3D points, its convex hull can be constructed in $O(1)$ expected number of passes with $O(n^{\epsilon})$ extra space, for any fixed $\epsilon > 0$.*

Remark: The algorithm can also be derandomized in the same bounds. Instead of making the set R random, we can use a $(1/B)$-*net* to make the set, where $B = O(n^{\epsilon})$. A streaming algorithm by Bagchi et al. [3] can deterministically compute this $(1/B)$-net in one pass with $O(\text{polylog}\, n)$ space.

3 Triangulation of Simple Polygons

In this section, we present our main result of the paper, which is on triangulating a simple polygon. This algorithm is not only based on the general techniques discussed in previous sections, but also based on new interesting properties of a type of special polygons proposed in Section 3.4.

More generally, our algorithm can triangulate an arbitrary set of disjoint line segments. Given a set L of n disjoint line segments in a plane, we show how to construct a triangulation of L (covering the convex hull of the endpoints of L and not using extra vertices) in $O(1)$ expected number of passes with $O(n^{\epsilon})$ extra space, for any fixed $\epsilon > 0$.

Before we describe our algorithm, we define some terms. By a *unimonotone* polygon, we mean an x-monotone polygon with one edge connecting its leftmost and rightmost vertex. We call this edge the *long* edge of the polygon.

Our algorithm consists of four major phases. In Section 3.1, we explain the construction of a trapezoidal decomposition of the line segments. In Section 3.2, we describe the transformation of the trapezoidal decomposition to a decomposition of unimonotone polygons. In Section 3.3, we describe the decomposition of

each unimontone polygons to a set of *special* polygons. In Section 3.4, we show how to triangulate a special polygon. In Section 3.5, we put these four phases together to obtain the overall algorithm.

3.1 Trapezoidal Decomposition of Line Segments

We construct the trapezoidal decomposition of a set of disjoint line segments recursively. This algorithm also uses the well-known random sampling approach [7,10]. The input L is a set of n disjoint line segments. The output T is the trapezoidal decomposition for T. We denote the set of line segments intersecting a trapezoid t by L_t. The parameter B will be determined later in this section.

Algorithm. Trap_Decomp(L)
If $|L| \leq B$
    Solve directly in memory
Repeat
    Randomly select a subset R of size B from L
    Build the trapezoidal decomposition T_R of R
Until $\sum_{t \in T_R} |L_t| = O(n)$ and $\max_{t \in T_R} |L_t| = O((n/B) \log B)$
For each $t \in T_R$
    T_t = Trap_Decomp(L_t)
Merge all the T_t's together to form T and return T

The set R is a randomly selected subset. Similar to analysis in Section 2, the first loop iterates only a constant expected number of times.

In the stream-sort model, we keep the set R and T_R in memory. By setting $B = n^\epsilon$, these two structures only take $O(n^\epsilon)$ space in memory. With T_R in memory, we can check the conditions to terminate the first loop in $O(1)$ passes.

We use the same the duplication idea used in Section 2. To prepare for the recursive calls in the second loop, in a scan pass, we create one copy of the segment for each trapezoid intersected and attach a label to each copy to identify the intersected trapezoid. In the sorting pass, we use the attached label as the key and group segments by the trapezoids intersected. Because $\max_{t \in T_R} |L_t| = O(n^{1-\epsilon}) \log n$, the number of levels of recursions is $O(1)$. Because $\sum_{t \in T_R} |L_t| = O(n)$, the intermediate stream for one level contains $O(n)$ line segments. Since there are only $O(1)$ levels of recursive calls, any intermediate stream contains $O(n)$ line segments.

The merging step can be done with one sorting pass and one scan pass. Each trapezoid is bounded by an upper edge, a lower edge, and two walls. In the sorting pass, we use the line segment of the upper edge as the primary key and the left-to-right order as the secondary key to sort all trapezoids. In the scan pass, trapezoids bounded by the same walls are all merged together.

We simultaneously solve subproblems in the same level in the same pass using the technique described in Section 1.1. Our algorithm can build the trapezoid decomposition in $O(1)$ rounds. Since the expected number of pass to obtain a valid trapezoid is $O(1)$, our algorithm takes $O(1)$ expected number of passes in total.

3.2 Decomposition of Line Segments into Unimonotone Polygons

In this section, we show how to construct a decomposition of line segments into unimonotone polygons. This is a well-known algorithm described in [8]. We only adapt it in the stream-sort model. The input T is a set of trapezoids from the trapezoidal decomposition described in the previous subsection. The output M is the set of unimonotone polygons from this decomposition.

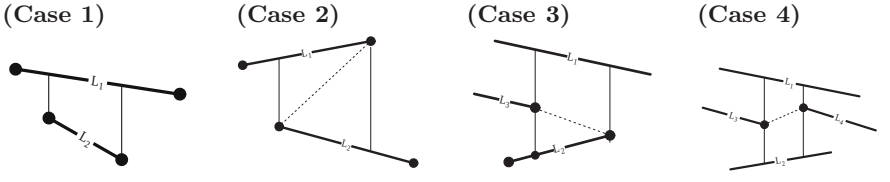


Fig. 1. Different cases to split a trapezoid

Algorithm. Monotone-Decomposition(T)
Initialize $S = \emptyset$
Check for each $t \in T$ //split trapezoids
  Case 1: the upper or lower edge of t is a whole segment
    Put t into S
  Case 2: two vertices are endpoints of line segments,
  but they are not from the same edge
    Draw the diagonal between the two vertices
    Split t into $t_\uparrow$ and $t_\downarrow$
    Put $t_\uparrow$ and $t_\downarrow$ into S
  Case 3: one vertex p is an endpoint of a line segment
  and another endpoint q of a line segment is on a vertical edge
    Draw the edge $\overline{pq}$
    Split t into a triangle Δ_t and trapezoid Q_t
    Put Δ_t and Q_t into S
  Case 4: no vertices are endpoints of line segments
  and two endpoints, p_1 and p_2 are on the vertical edges
    Draw the edge $\overline{p_1 p_2}$
    Split t into two trapezoids T_1 and T_2 and put T_1 and T_2 into S
For each line segment l //merge polygons
  Sort all polygons using l as the upper edge from left to right
  Merge all these polygons to form a unimonotone polygon m
  Put m into M
  Do the same for all polygons using l as the lower edge
Return M

Now we modify both steps of the algorithm for the stream-sort model. Step 1 can be simply done by a scan pass. Instead of writing the results to a data structure S, we write them into the output stream. For step 2, in a sorting pass, we use

the segment of the upper edge as the primary key and the left-to-right order as the secondary key to group and sort polygons. In a scan pass, we merge the polygons, whose upper edges are of the same segment, to a unimonotone polygon and write the polygon to the output stream. We do the same for the polygons whose lower edges are of the same segment. Both of these two steps take $O(1)$ passes with $O(1)$ extra space.

3.3 Decomposition of a Unimonotone Polygon into Special Polygons

It is not obvious how to triangulate unimonotone polygons directly in the stream-sort model. In this section and the next, we introduce nontrivial new ideas that differ from the approaches in previous (sequential or parallel) polygon triangulation algorithms. The following definition is the key:

Definition 1. *Given a direction d, a unimonotone polygon is a* special *polygon at direction d, if its chain of edges is monotone in direction d and both vertices of the long edge are higher than any other vertices in the direction perpendicular to d.*

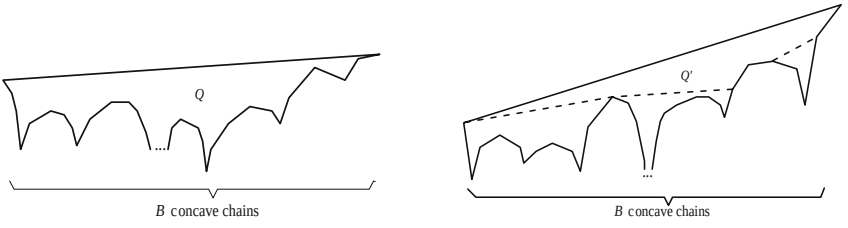


Fig. 2. Decomposition of Q into special polygons

The decomposition is built recursively. Given the unimonotone polygon P, we divide the monotone chain into B parts with equal size, and build the upper hull for each part. Below the upper hulls, the decomposition will be built recursively. Above the upper hulls, we obtain a new unimonotone polygon, whose monotone chain Q is formed by at most B concave chains. See Fig. 2. The following top-down sweeping algorithm decomposes Q into a set of special polygons. Interestingly, this part is inspired by a well-known algorithm for constructing the 2D maxima of a set of points [11]. The input Q is an x-monotone chain formed by B concave chains. The output C is the set of edges of the special polygons. Without loss of generality, we assume the right end is higher than the left end.

Algorithm. Special_Decomp(Q)
Initialize v to the right end of the long edge and C empty
Put all vertices along the y-direction in a sorted list L in decreasing order
For each vertex v_i in decreasing y-order
    If v_i is left of v

Add edge (v_i, v) to C
$v = v_i$
Return C

By this algorithm, we connect the right endpoint of the long edge to the left end with an xy-monotone chain, and all regions below this chain are special polygons along the x-direction, as one can easily see. Consider the region Q' above the xy-monotone chain. (See Fig. 2.) This is also a special polygon, where, this time, the direction d is perpendicular to the long edge. In the scan pass, we only need to keep v and v_i in memory. Any created edge in C is written into the output stream as scan proceeds. With a sorting pass, we use sort all edges using the x-coordinate of the first point as the primary key and the x-coordinate of the second point as the secondary key. Then, with another scan pass, we can obtain the xy-monotone chain in order. Therefore, in the stream-sort model, the above algorithm takes $O(1)$ passes with $O(1)$ space in memory.

By setting B to n^ϵ, the number of levels of the recursion is $O(1)$. Then we simultaneously solve all problems in the same level of the recursion tree in one round. Because any edge we write to the output stream is an edge in the final triangulation and the size of the triangulation is $O(n)$, the size of all intermediate streams is $O(n)$. Since the upper hulls are computed by our algorithm in Section 2, it takes $O(B)$ space in memory with $O(1)$ passes.

3.4 Triangulation of a Special Polygon

We build a triangulation for a special polygon, say in the direction of the x-axis, by a top-down sweeping procedure. In this algorithm, we maintain a tree structure *bridges* in memory. It stores a set of pairs *(left, right)*. Each pair corresponds to one portion of the chain intersecting the sweepline. They are ordered left to right in direction y. For p, which is a query point or a pair in *bridges*, its predecessor and successor in *bridges* refer to the bridge immediately left of and right of p, denoted as p^- and p^+, respectively. There are two types of events: 1. the sweepline touches a vertex which does not belong to an edge already intersecting the sweepline and 2. the sweepline touches a vertex which belongs to an edge already intersecting the sweepline. The input p is a special polygon whose monotone chain is monotone in x direction without loss of generality. For any other special polygon in other directions, we can rotate the coordinate plane, so that the polygon is a special polygon in direction of y-axis.

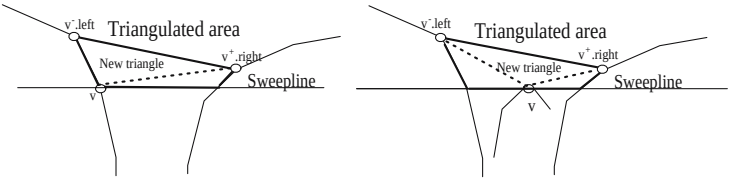


Fig. 3. Adding a new triangle into the triangulated area

Algorithm. SpecialTriangulate(P)
Put all vertices on the monotone chain top-down in direction y in a sorted list L
Let the left end point of the vertex be E_l and the right one be E_r
Initialize *bridges* with two pairs $(-\infty, E_l)$ and (E_r, ∞)
While L is not empty
    Pick the next vertex v from L
    Add triangle $(v^-.\text{right}, v, v^+.\text{left})$ to T
    Case 1: v causes an event of the type 1 // see Fig. 3 (right)
        Add (v, v) to *bridges*
    Case 2: v causes an event of the type 2 // see Fig. 3 (left)
        If the left side of v along the sweepline is inside P
            $v^+.\text{left} = v$
        Else if the right side of v along the sweepline is inside P
            $v^-.\text{right} = v$
        Else
            Merge v^- and v^+ to $(v^-.\text{left}, v^+.\text{right})$

The proof of the correctness is presented in the full version of this paper.

In the stream-sort model, we use a sorting pass to prepare L. Then we perform the loop part of the algorithm and write all triangles into the output stream in one scan pass, and store only *bridges* in memory. Therefore the extra space used in memory is linear to the maximum number of edges intersecting the sweepline.

Lemma 1. *Given a special polygon at direction d, it can be triangulated in $O(1)$ passes with $O(m)$ extra space, where m is the maximum number of edges of the polygon intersected by a line in direction d.*

3.5 Triangulation of Line Segments

By solving multiple subproblems in one round, we simultaneously construct the triangulation of all special polygons decomposed from the same level of the recursive calls in the monotone decomposition algorithm described in Section 3.3. Recall that the special polygons are decomposed from a unimonotone polygon whose monotone chain is composed of n^ϵ upper hulls. We conclude that the special polygon Q' above the xy-monotone chain and all obtained special polygons below the xy-monotone chain (see Fig. 2) have $O(n^\epsilon)$ edges intersecting its sweepline. The extra space used to triangulate one special polygon is $O(n^\epsilon)$. Note that all of the four phases take $O(1)$ passes with $O(n^\epsilon)$ space.

Theorem 2. *A triangulation of a set of disjoint line segments can be constructed in $O(1)$ expected number of passes with $O(n^\epsilon)$ extra space, for any fixed $\epsilon > 0$.*

Remark: We also can adapt the trapezoidal decomposition algorithm to report intersections for a set of line segments with intersections. We modify the random sampling verification step used in the algorithm in Section 3.1. Here we use $\sum_{t \in T_R} |L_t| \leq c(n + kB/n)$ [7], where k is the number of intersections detected

so far. If this is a valid random sample, we build the trapezoidal decomposition on this sample, break the segment intersecting the boundary of the trapezoidal decomposition into two segments, one above the boundary and one below the boundary.

4 Constructing the 2D Convex Hull of x-Ordered Points in a Weaker Streaming Model

The sorting operation is expensive in the stream-sort model. We here describe how to construct the 2D convex hull of a set of x-ordered points in $O(1)$ passes with $O(n^\epsilon)$ extra space, in the direction-flexible model. This result also contrasts with the near-$\sqrt{n}$ lower bound result for the same problem given by Chan and Chen [6] in the original multi-pass stream model.

We only describe how to construct the upper hull, because constructing the lower hull is symmetric. Computing the lower hull simultaneously at most doubles the size of the intermediate stream.

Given a set P of 2D points and a vertical line h, we define the *bridge* at h as the edge of the upper hull of P intersecting h. To compute the bridge at h, we transform the problem in dual space. In dual space, each point becomes a halfplane and a bridge corresponds to an extreme point in the intersection of halfplanes. This point can be computed using linear programming. With $O(n^\epsilon)$ extra space in memory, this linear programming problem can be solved under the multi-pass model in $O(1)$ passes by the results of Chan and Chen [6].

We find all points not on the upper hull by divide-and-conquer. The input is a set P of 2D points sorted in x-order. In the output, all points not on the upper hull are marked. In the final pass, we scan this output, and report all unmarked points in x-order, which form the upper hull of P.

Algorithm. MarkUH(P)
If $|P| = O(Bn^\epsilon)$
    Solve the problem directly in memory
Else
    Divide P into B groups: $P_1, P_2, ..., P_B$
    Find the set H of vertical lines between any two adjacent groups
    For P, compute the bridges at each $h \in H$.
    Mark any $p \in P$ if p is under one of these bridges
    For each P_i
        MarkUH(P_i)

Again, we solve all subproblems in one level simultaneously. For each subproblem, we need to compute $B - 1$ bridges. The computation for each bridge corresponds one linear-programming problem. We use the multi-pass linear-programming algorithm mentioned above. This algorithm scans an array of data sequentially multiple times without modifying it, and the order of the scan does not affect the algorithm. However, between scans, addition information must be kept by the algorithm in memory.

In each subproblem, we compute all $B-1$ bridges by simulating this multi-pass linear programming algorithm. The memory contents of these simulations are kept in memory. After all data of one subproblem are scanned, we write all memory contents to the output stream after the data of that subproblem. Then the memory is reset for the next subproblem in the stream. Thus, in intermediate streams, data and memory contents of the multi-pass algorithm are interleaved. For the next scan, this output stream becomes the input stream. We start from the other end of the stream, so that we can reload the memory content of each subproblem back in memory, before any data point of that subproblem is accessed. After the final pass, all bridges are determined. They are written into the stream after the corresponding data set. To mark points under a bridge, we start from the other end. All bridges are loaded into memory, before the corresponding data set is accessed. This step takes $O(1)$ passes. By setting $B = O(n^\epsilon)$, the height of the tree is constant. Therefore, the total number of passes is $O(1)$ and the memory required is $O(n^{2\epsilon})$.

Besides one mark for each item, all extra information written into the stream consists of the memory contents produced by the multi-pass algorithm. At each level, the total number of bridges to compute is at most $O(\frac{n}{Bn^\epsilon})$. The total extra space is $O(n/B) = o(n)$. By setting $2\epsilon = \delta$, we have:

Theorem 3. *Given a set of x-ordered 2D points, its convex hull can be constructed in $O(1)$ passes with $O(n^\delta)$ extra space in the direction-flexible streaming model, for any fixed $\delta > 0$.*

Acknowledgement. I thank Timothy Chan for suggesting the topic of geometric algorithms in the stream-sort model, and for help in the revision of the manuscript.

References

1. Aggarwal, A., Chazelle, B., Guibas, L., Ó'Dúnlaing, C., Yap, C.: Parallel computational geometry. Algorithmica 3 1, 293–327 (1988)
2. Aggarwal, G., Datar, M., Rajagopalan, S., Ruhl, M.: On the streaming model augmented with a sorting primitive. In: Proc. of the 45th Annual IEEE Symposium on Foundations of Computer Science, pp. 540–549 (2004)
3. Bagchi, A., Chaudhary, A., Eppstein, D., Goodrich, M.T.: Deterministic sampling and range counting in geometric data streams. In: Proc. of 20th ACM Annual Symposium on Computational Geometry, pp. 144–151 (2004)
4. Brönnimann, H., Chan, T.M., Chen, E.Y.: Towards in-place geometric algorithms and data structures. In: Proc. of 20th ACM Annual Symposium on Computational Geometry, pp. 239–246 (2004)
5. Chan, T.M.: Faster core-set constructions and data-stream algorithms in fixed dimensions. Computational Geometry: Theory and Applications 35, 20–35 (2006)
6. Chan, T.M., Chen, E.Y.: Multi-pass geometric algorithms. In: Proc. of 21st ACM Annual Symposium on Computational Geometry, pp. 180–189 (2005)
7. Clarkson, K.L., Shor, P.W.: Applications of random sampling in computational geometry, ii. Discrete and Computational Geometry 4(1), 387–421 (1989)

8. de Berg, M., van Kreveld, M., Overmars, M., Schwarzkopf, O.: Computational Geometry, Algorithms and Applications. Springer, Heidelberg (1997)
9. Lammersen, C., Sohler, C.: Strsort algorithms for geometric problems. In: European Woarkshop on Computatonal Geometry, pp. 69–72 (2007)
10. Mulmuley, K.: Computational Geometry, An Introduction Through Randomized Algorithms. Prentice Hall, Englewood Cliffs (1994)
11. Preparata, F.P., Shamos, M.I.: Computational Geometry: An Introduction. Springer, Heidelberg (1985)
12. Ruhl, J.M.: Efficient Algorithms for New Computational Models. PhD thesis, Massachusetts Institute of Technology (2003)

External Memory Range Reporting on a Grid

Yakov Nekrich

Dept. of Computer Science, University of Bonn
yasha@cs.uni-bonn.de

Abstract. In this paper we present external memory data structures for orthogonal range reporting queries on a grid. Our data structure for two-dimensional orthogonal range reporting queries uses $O((N/B)\log_2 N)$ blocks of space of size B and supports queries in optimal $O(\log_2 \log_B U + T/B)$ time, where U is the size of universe, N is the number of elements in the data structure, and T is the size of the answer. Our data structure for three-sided range reporting queries that uses $O(N/B)$ blocks of space and supports queries in $O(\log_2 \log_B U + T/B)$ time. In the case of three-sided range reporting on a $N \times \mathbb{N}$ grid, we describe a $O((N/B)\log_B^2 N)$ space data structure with $O(T/B)$ query time, a $O((N/B)\log_B^* N)$ data structure with $O(\log_B^* N + T/B)$ query time, and a $O(N/B)$ space data structure with $O(\log_B^{(k)} N + T/B)$ query time for any constant k.

1 Introduction

In the I/O model the data is stored in *disk blocks* of size B, a block can be read into internal memory from disk (resp. written from internal memory into disk) with one I/O operation, and computation can only be performed on data stored in the internal memory. We refer to e.g. [22] or [1] for a detailed description of the I/O model and its variants. The I/O complexity of different data structures was studied extensively during the last decades (see e.g. [22], [5] for surveys). In the comparison model, when only comparisons between elements in the internal memory are allowed, $\Omega(\log_B N)$ is a natural lower bound for many data structure problems of size N. In this paper we show that the $\Omega(\log_B N)$ barrier can be surpassed for range reporting queries in the case when the universe size is bounded (say, by a polynomial function of the number of elements) and other operations besides comparisons are allowed.

A number of searching problems can be solved more efficiently in the different variants of the internal-memory RAM model when the size of the universe is bounded, i.e. when the elements (resp. point coordinates) are integers in the range $[0, U-1]$ for an appropriate U. The well known example is the van Emde Boas data structure [11] that supports predecessor queries in $O(\log_2 \log_2 U)$ time and uses linear space. There are efficient data structures for several geometric problems: data structures described by Overmars [17] and Alstrup *et al.* [2] support two-dimensional orthogonal range reporting queries in $O(\log_2 \log_2 U + T)$ time, where T is the size of the answer, three-dimensional orthogonal range reporting queries can be answered in $O(\log_2 \log_2 U + (\log_2 \log_2 N)^2 + T)$ time [16],

T. Tokuyama (Ed.): ISAAC 2007, LNCS 4835, pp. 525–535, 2007.

point location in two- and three-dimensional rectangular subdivisions can be answered in $O(\log_2 \log_2 U)$ and $O((\log_2 \log_2 U)^2)$ time respectively using the data structure of de Berg *et al.* [8]; general two-dimensional point location queries can be answered in $O(\log_2 \log_2 U)$ time using the data structure of Amir *et al.* [3], but the space usage can be very high in the worst case. Recently, Chan [9] and Pătraşcu [18] independently presented a linear space data structure for two-dimensional point location queries with $O(\sqrt{\log_2 U / \log_2 \log_2 U})$ query time.

External memory data structures for three-sided range reporting and two-dimensional range reporting were studied in a number of papers, e.g. [14], [20], [21], [13], [6]. Arge *et al.* [6] present a data structure that supports two-dimensional orthogonal range reporting queries in $O(\log_B N + T/B)$ query time, where T is the number of points in the answer, and uses $O((N/B)\log_2 N/\log_2 \log_2 B)$ space. The space usage of the data structure of [6] is optimal in certain computational models [21], [6]. In the same paper [6] the authors also describe a linear space data structure for three-sided queries with $O(\log_B N + T/B)$ query time. A linear space data structure of [13] supports general two-dimensional range queries in $O(\sqrt{N/B} + T/B)$ time.

In this paper we present for the first time external memory data structures that support range reporting queries on a $U \times U$ grid. We describe a data structure for orthogonal range reporting queries with $O(\log_2 \log_B U + T/B)$ query time and $O((N/B)\log_2 N)$ space. According to the lower bound for predecessor queries of [19] and the reduction of two-dimensional emptiness queries to predecessor queries [15], our data structure achieves the optimal query time. Our data structure for three-sided range reporting on a grid uses $O(N/B)$ blocks of space and supports queries in $O(\log_2 \log_B U + T/B)$ time, and thus achieves optimal space and query time. We also consider the problem of three-sided range reporting on $N \times \mathbb{N}$ grid, i.e., the x-coordinates belong to $[1, N]$ and y-coordinates are arbitrary integers. We show that $O(T/B)$ query time can be achieved with $O((N/B)\log_B^2 N)$ space data structure. Two other space-time trade-offs for three-sided queries on $N \times \mathbb{N}$ grid are also described: the $O((N/B)\log_B^* N)$ space data structure supports queries in $O(\log_B^* N + T/B)$ time and $O(N/B)$ space data structure supports queries in $O(\log_B^{(k)} N + T/B)$ query time [1] for an arbitrary integer constant k. Our data structures for three-sided queries on $N \times \mathbb{N}$ grid use only comparisons and indirect addressing. Data structures for queries on a $U \times U$ grid use multiplications and divisions. If the set of allowed operations is limited by comparisons, indirect addressing, additions, and bit shifts, then queries on a $U \times U$ grid can be supported in $O(\sqrt{\log_B U} + T/B)$ time, see Theorem 2.

Throughout this paper N denotes the number of elements in the data structure, T denotes the number of elements in the answer, B denotes the disk block size, and U denotes the size of the universe. In this paper we use the number of I/O operations as the time measure; the space usage of the data structures is measured in the number of disk blocks.

[1] We define $\log_B^* N = \min\{k | \log_B^{(k)} N \leq B\}$, where $\log_B^{(1)} N = \log_B N$ and $\log_B^{(k)} N = \log_B(\log_B^{(k-1)} N)$ for $k > 1$.

2 Preliminaries

Let the predecessor and the successor of an element x in the set S be defined as $pred(x, S) = \max\{e \in S | e \leq x\}$ and $succ(x, S) = \min\{e \in S | e \geq x\}$. The following result is shown in [19], for completeness we provide a sketch of the proof here.

Statement 1. *Given a set $S \subset [1, U]$, there exists a $O(N/B)$ space data structure for S that supports predecessor and successor queries in $O(\log_2 \log_B U)$ time.*

Proof. This result can be achieved by a combination of sub-sampling with the van Emde Boas data structure [11]. A data structure that uses $O(N)$ blocks of space can achieve $O(\log_2 \log_B N)$ query time by applying the van Emde Boas approach. The van Emde Boas data structure enables us to reduce the size of the universe from U to $\sqrt{U}$ in $O(1)$ time. Hence, in $O(\log_2(\log_2 U/\log_2 B))$ time the size of the universe can be reduced to B. Clearly, a predecessor query in a universe of size B can be answered in constant time with a $O(1)$ space data structure. There are $O(N)$ such data structures; hence, this construction uses $O(N)$ blocks of space. Finally, a $O(N)$ space data structure can be turned into $O(N/B)$ data structure using sub-sampling. The set S is divided into $S_1, S_2, \ldots, S_{N/B}$, such that each S_i contains B consecutive elements of S and each element in S_i is smaller than each element in S_j for $i < j$. The set S' contains one element from each S_i. The predecessor e' of x in S' can be found in $O(\log_2 \log_B N)$ time with $O(N/B)$ space data structure. The predecessor of e in S is found in $O(1)$ time by searching in the group S_i that corresponds to e'.

In the case when the set of possible operations is limited by comparisons, additions, indirect addressing, and bit shifts, the following data structure can be constructed

Statement 2. *Given a set $S \subset [1, U]$, there exists a $O(N/B)$ space data structure for S that supports predecessor and successor queries in $O(\sqrt{\log_B U})$ time and uses only comparisons, additions, indirect addressing, and bit shifts.*

Proof. This data structure is a modification of the data structure of [23] for the external memory. The set S is divided into subsets S_i of size $B^{\sqrt{\log_B U}+1}\sqrt{\log_B U}$ each, so that every element in S_i is smaller than every element in S_j for $i < j$. Let S' be the set that contains exactly one representative element for each subset S_i. We construct a trie $\mathcal{T}_t$ with node degree $B^{\sqrt{\log_B U}}$ for the elements of S. Each trie node uses $O(B^{\sqrt{\log_B U}-1})$ disk blocks. The height of the trie is $O(\sqrt{\log_B U})$. S' contains $O(N/(B^{\sqrt{\log_B U}+1}\sqrt{\log_B U}))$ elements; hence, the total number of nodes is $O(N/B^{\sqrt{\log_B U}+1})$. Therefore, the trie for S' can be stored in $O(N/B)$ blocks. For each S_i we construct a B-tree $\mathcal{T}_i$. All B-trees also use $O(N/B)$ blocks of space.

To find $pred(x, S)$ for some x, we find $e' = pred(x, S')$ in $O(\sqrt{\log_B U})$ time using the trie $\mathcal{T}_t$. Clearly, $pred(x, S)$ belongs either to S_i or to S_{i+1} , where S_i

is the set that contains e'; hence, $pred(x, S)$ can be found in $O(\sqrt{\log_B U})$ time with help of $\mathcal{T}_i$ or $\mathcal{T}_{i+1}$.

The data structure of Statement 2 can be dynamized, although this is not relevant for the further exposition. Details of the dynamic data structure will be given in the full version of this paper.

3 Three-Sided Range Reporting on $N \times \mathbb{N}$ Grid

Three-sided queries are a special case of two-dimensional orthogonal range queries when the query range Q is a product of a closed interval $[a, b]$ and a half-open interval $[c, +\infty)$. Dominance queries are a special case of three-sided queries when the query range ia a product of two half-open intervals. In this section we describe three fast data structures for three-sided range reporting queries on $N \times \mathbb{N}$ grid, i.e. in the case when the x-coordinates of points are bounded by N and the y-coordinates are arbitrary integers. A three-sided query on a $U \times \mathbb{N}$ grid can be reduced to a three-sided range query on a $N \times \mathbb{N}$ grid using the reduction to rank space technique [10]. Data structures for range reporting on a $U \times U$ grid will be described in section 4.

Lemma 1. *There exists a data structure A that uses $O(\frac{N}{B} \log_B^2 N)$ blocks of space and supports three-sided range reporting queries in $O(T/B)$ time.*

Proof. The main idea of our approach is the search of *the leaf-to-root paths* of the external memory priority search tree [6]. A similar approach was also used in the internal memory data structure of Fries *et al.* [12]. However, in our construction we show that a three-sided query can be answered by answering one-dimensional queries to certain data structures for leaf-to-root paths; this allows us to achieve constant query time.

We construct a B-tree $\mathcal{T}_x$ on the set of all possible x-coordinates, i.e all integers in $[0, N-1]$ are stored in the leaves of $\mathcal{T}_x$. For a leaf x of $\mathcal{T}_x$, π_x is the path from x to the root of $\mathcal{T}_x$. For a path π_x we define two sets of nodes: π_x^+ and π_x^-. For every internal node v that belongs to π_x, π_x^+ contains nodes $v_{j+1}, v_{j+2} \ldots, v_B$ of $\mathcal{T}_x$, where v_j is a child of v that belongs to π_x and $v_{j+1}, \ldots, v_B$ are the children of v that follow v_j. Analogously, for every internal node v on π_x π_x^- contains all nodes $v_1, \ldots, v_{j-1}$, where v_j is a child of v that belongs to π_x and $v_1, \ldots, v_{j-1}$ are the children of v that precede v_j. Observe that all π_x^+ and π_x^- contain $O(B \log_B N)$ nodes. For a node v, $lev(v)$ is the number of edges between v and a leaf of $\mathcal{T}_x$. Let $\pi_a^+(q)$ and $\pi_b^-(q)$ denote the sets of nodes on levels $1, 2, \ldots, lev(q) - 2$ that belong to π_a^+ and π_b^- respectively. For simplicity we will sometimes say that a data structure supports range reporting queries in constant time if the query time is $O(T/B)$, where T is the number of points in the answer.

We say that a point p belongs to an internal node v if it is stored in a leaf descendant of v. We construct a set Y_v for every internal node v of $\mathcal{T}$ starting with the root. Y_v consists of B points with maximal y-coordinates among all points that belong to v and are not stored in sets Y_w for any ancestor w of v.

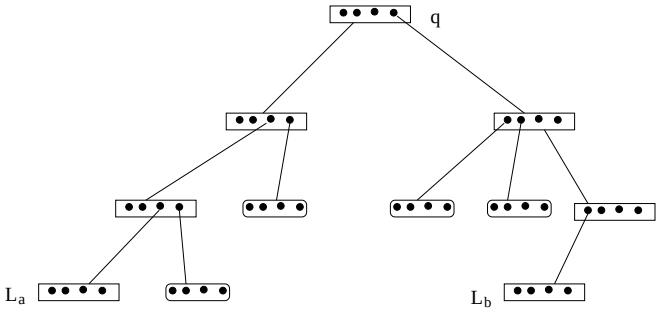


Fig. 1. An illustration of the algorithm for three-sided range reporting. Points with x-coordinates a and b are stored in leaves L_a and L_b. The lowest common ancestor q of L_a and L_b, and all nodes on π_a and π_b that are descendants of q are drawn as rectangles. All nodes on $\pi_a^+(q)$ and $\pi_b^-(q)$ are drawn as ovals.

We denote by $\min_v$ the point with minimum y-coordinate stored in Y_v and say that $\min_v$ *represents* the node v. For every leaf-to-root path π_x and each node $q \in \pi_x$ we construct four data structures: $S_x^+(q)$, $S_x^-(q)$, $M_x^+(q)$, and $M_x^-(q)$. Data structure $S_x^+(q)$ contains elements of sets Y_v for all nodes v that belong to $\pi_x^+(q)$. Data structure $S_x^-(q)$ contains all elements of sets Y_v for all nodes v that belong to $\pi_x^-(q)$. $M_x^+(q)$ ($M_x^-(q)$) contains points $\min_v$ for all nodes v that belong to $\pi_x^+(q)$ ($\pi_x^-(q)$). For each node q on every path π_x there are data structures $G_x^+(q)$ and $G_x^-(q)$: $G_x^+(q)$ ($G_x^-(q)$) contains all points p, such that p belongs to some node u on π_x^+ (π_x^-) on levels $1, 2, \ldots, lev(q) - 2$ and p is stored in a set Y_w for some ancestor w of u. Points in all $S_x^+(q)$, $S_x^-(q)$, $M_x^+(q)$, $M_x^-(q)$, $G_x^+(q)$ and $G_x^-(q)$ are stored in the descending order of their y-coordinates, so that one-dimensional queries $[c, +\infty)$ can be answered in constant time. In every node v we store data structure D_v that contains all elements of Y_{v_i} for all children v_i of v and supports three-sided range reporting queries. Additionally, in every node v there is a data structure F_v: for all ancestors w of v, F_v contains information about those points in Y_w that belong to the node v. For every point $p = (x, y)$ that is stored in some Y_w and belongs to v we store the point (i, y) in F_v, where i is such that p belongs to the i-th child v_i of v. We will show later in this section how F_v can support three-sided queries in constant time.

Given a query $[a, b] \times [c, +\infty)$, let q denote the lowest common ancestor of the leaves L_a and L_b that contain a and b. Let q_a and q_b be the children of q that belong to π_a and π_b respectively. If the x-coordinate of a point p is contained in the interval $[a, b]$, then one of the following three conditions is satisfied: 1. p belongs to a node v on π_a^+ or a descendant of a node on π_a^+. 2. p belongs to a node v on π_b^- or a descendant of a node on π_b^-. 3. p belongs to one of the nodes $q_{a+1}, q_{a+2}, \ldots, q_{b-1}$ (i.e. to one of the children of node q between q_a and q_b). Hence the search procedure must examine: 1. Points stored in sets Y_v where v is a node that belongs to $\pi_a^+(q)$; if necessary, the sets Y_w for some descendants w of the nodes that belong to $\pi_a^+(q)$ are also examined. 2. Points stored in sets Y_v where v is a node that belongs to $\pi_b^-(q)$; if necessary, the sets

Y_w for some descendants w of the nodes that belong to $\pi_b^-(q)$ are also examined. 3. Points stored in the sets $Y_{q_{a+1}}, Y_{q_{a+2}}, \ldots, Y_{q_{b-1}}$, where $q_{a+1}, q_{a+2}, \ldots, q_{b-1}$ are the children of q between q_a and q_b; if necessary, the sets Y_w for some descendants w of $q_{a+1}, q_{a+2}, \ldots, q_{b-1}$ are also examined. 4. Some points that belong to one of the nodes $q_{a+1}, \ldots, q_{b-1}$ can be stored in a set Y_w for some ancestor w of q; such points can be tested with help of the data structure F_q. Some points that belong to one of the nodes v, $v \in \pi_a^+(q)$ ($v \in \pi_b^-(q)$) may be stored in a set Y_u for some ancestor u of v; such points can be tested with help of the data structure $G^+_{\pi_x,q}$ ($G^-_{\pi_x,q}$). In every set of points examined by the above search procedure we must output all points whose y-coordinates are not smaller than c.

All points with y-coordinates not smaller than c stored in sets Y_v, where v is a node that belongs to $\pi_a^+(q)$, can be found by a query $[c, +\infty)$ to $S_a^+(q)$. We can decide which nodes of the $\pi_a^+(q)$ must be visited by the same query $[c, +\infty)$ to $M_a^+(q)$. In every visited node w we use D_w to report all points in the sets $Y_{w_1}, Y_{w_2}, \ldots, Y_{w_B}$, where $w_1, w_2, \ldots w_B$ are the children of w, whose y-coordinates are at least c. If all B points in some set Y_{w_i} have y-coordinates that are greater than or equal to c, then the node w_i is also visited. In every visited descendant u of w, we use D_u to identify points whose y-coordinates are greater than or equal to c that were not output when the ancestors of u were processed. Then we recursively visit all children u_i of u, such that Y_{u_i} contains B points whose y-coordinates are greater than or equal to c. The points that belong to the query range and are stored in some set Y_v, where v belongs to $\pi_b^-(q)$, or in some set Y_w, where w is a descendant of a node that belongs to $\pi_b^-(q)$, can be found with the same procedure. Finally, we identify all points in $Y_{q_{a+1}}, Y_{q_{a+2}}, \ldots, Y_{q_{b-1}}$ whose y-coordinates are not smaller than c using a corresponding three-sided query to D_q and visit the children q_i of q, such that $a < i < b$ and Y_{q_i} contains B points with y-coordinates equal to or exceeding c. In every visited node q_i we identify the points stored in D_{q_i} with y-coordinate at least c, and we recursively visit such descendants w of q_i that Y_w contains B points whose y-coordinates are greater than or equal to c. Some of the points that belong to children $q_{a+1}, \ldots, q_{b-1}$ of q may be stored in sets Y_u for some ancestor u of q. Such points, if their y-coordinates are not smaller than c, can be reported with a query $[a+1, b-1] \times [c, +\infty)$ to F_q. Some of the points that belong to some node v on $\pi_a^+(q)$ ($\pi_b^-(q)$) may also be stored in sets Y_u for some ancestor u of v. The relevant points can be found with a query $[c, +\infty)$ to $G^+_{\pi_a,q}$ ($G^-_{\pi_b,q}$). It remains to test the points that are stored in the set Y_q. Since Y_q contains $O(B)$ elements, all points in Y_q that are contained in the query range $[a, b] \times [c, +\infty)$ can be reported in $O(1)$ time.

Since every point is stored in one set Y_v, all data structures D_v use $O(N/B)$ blocks of space. The data structure F_v with f elements uses $O(\max(1, f/B))$ blocks of space. Since every point occurs in $O(\log_B N)$ data structures and there are $O(N/B^2)$ internal nodes of $\mathcal{T}_x$, all data structures F_v use $O((N/B)\log_B N)$ blocks of space. Each data structure $S_x^+(v)$, $S_x^-(v)$, $G_x^+(v)$ or $G_x^-(v)$ for a path π_x uses $O(B \log_B N)$ blocks; hence, all data structures $S_x^+(v)$, $S_x^-(v)$, $G_x^+(v)$ and $G_x^-(v)$ for all nodes on all paths π_x use $O(NB \log_B^2 N)$ blocks of space. All data

structures $M_x^+(v)$, and $M_x^-(v)$ use $O(N \log_B^2 N)$ blocks of space. To sum up, all data structures associated with all paths π_x use $O(NB \log_B^2 N)$ blocks, and all data structures associated with all internal nodes v of $\mathcal{T}_x$ use $O((N/B) \log_B N)$ blocks.

The space usage can be reduced to $O((N/B) \log_B^2 N)$ using a standard subsampling technique. We subdivide $[1, N]$ into the intervals of size B^2. The set P'_x contains one representative element of $P_x \cap I_j$ for each interval I_j, where $I_j = [B^2(j-1)+1, B^2 j]$ and P_x is the set of x-coordinates of all points in P. All points whose x-coordinates belong to an interval I_j are stored in a data structure C_j; C_j supports three-sided queries in $O(1)$ time because I_j contains $O(B^2)$ points [6]. Data structures $S_x^+(q)$, $S_x^-(q)$, $M_x^+(q)$, $M_x^-(q)$, $G_x^+(q)$ and $G_x^-(q)$ are stored only for those leaves of $\mathcal{T}_x$ that correspond to elements of P'_x. Let $a' = succ(a, P'_x)$, $b' = pred(b, P'_x)$. To answer a query $Q = [a, b] \times [c, +\infty)$, we report all points in $[a, a'-1] \times [c, +\infty)$ using C_a, all points in $[b'+1, b] \times [c, +\infty)$ using C_b, and if $a' \neq b'$, we report all points in $[a', b'] \times [c, +\infty)$ using data structures for $\pi_{a'}$ and $\pi_{b'}$ as described above.

In the above description we assumed that all points have different x-coordinates. The case when many points can have the same x-coordinates can be dealt with as follows. Let P' be the set that contains (at most) B points with maximal y-coordinates for any possible x-coordinate x. All points of P with the same x-coordinate x are stored in the list L_x sorted by their y-coordinates. All lists L_x can be packed in $O(N/B)$ blocks of space. The points of the set P' may be stored in the data structure described above with slight modifications: we associate two blocks of space with every leaf of $\mathcal{T}_x$, so that all points with the same x-coordinate are stored in one leaf, and the number of points stored in every leaf is at least B. All data structures for nodes and leaf-to-root paths of $\mathcal{T}_x$ are constructed in the same way as above. All points $(x, y) \in P$, such that there are at most B points with x-coordinate x that belong to the query range $[a, b] \times [c, +\infty)$ can be found with a query to P'. Let P'' be the set that contains for every $x \in [0, N-1]$, such that P' contains B points with x-coordinate x, the point with the minimum y-coordinate among all points of P' whose x-coordinate equals to x. All points of P'' have different x-coordinates; hence, the same data structure as described above can be used. For every point $(x, y) \in P''$ that belongs to the query range we examine the corresponding list L_x and report all points whose y-coordinate is at least c. All relevant lists L_x can be examined in $O(T/B)$ time. Hence, all points $(x, y) \in P$ such that there are more than B points with x-coordinate x that belong to $[a, b] \times [c, +\infty)$ can be reported in $O(T/B)$ time.

To complete the proof, it remains to describe how the data structure F_v is implemented. Let P be the set of points in F_v, and let P', P'', and L_x be defined as in the previous paragraph. Since F_v contains $O(B)$ points with different x-coordinates, P' and P'' contain $O(B)$ and $O(B^2)$ points respectively, and three-sided queries on P' and P'' can be supported in constant time. Hence, queries on P can be also supported in constant time in the same way as described in the previous paragraph.

We can obtain two furher space-time trade-offs for three-sided queries on $N \times \mathbb{N}$ grid. These results are obtained by an application of the bootstrapping paradigm to the slightly modified construction of Lemma 1.

Lemma 2. *Given a data structure A that uses $s(N)$ blocks of space and supports three-sided range reporting queries on $N \times \mathbb{N}$ grid in $q_1(N) + O(T/B)$ time and dominance reporting queries on $N \times \mathbb{N}$ grid in $q_2(N) + O(T/B)$ time. Then there is also a data structure D that uses $s(N) + O(\frac{N}{B})$ blocks of space and supports three-sided range reporting queries on $N \times \mathbb{N}$ grid in $\max(q_1(\log_B N), 2q_2(\log_B N)) + O(1) + O(T/B)$ time and dominance reporting queries on $N \times \mathbb{N}$ grid in $q_2(\log_B N) + O(1) + O(T/B)$ time.*

Proof. The construction is alomst the same ass in the proof of Lemma 1, but sub-sampling is applied in a different way. We subdivide $[1, N]$ into the intervals of size $B^2 \log_B^2 N$. The set P'_x contains one representative element of $P_x \cap I_j$ for each interval I_j where $I_j = [(j-1)B^2 \log_B^2 N + 1, jB^2 \log_B^2 N]$. All points whose x-coordinates belong to I_j are stored in an interval data structure C_j that supports three-sided queries in $O(q(\log_B N) + T/B)$ time. Again, data structures $S_x^+(q)$, $S_x^-(q)$, $M_x^+(q)$, $M_x^-(q)$, $G_x^+(q)$ and $G_x^-(q)$ are stored only for those leaves of $\mathcal{T}_x$ that correspond to elements of P'_x. Besides that, data structures F_v are stored only for the nodes v that belong to some path π_x, $x \in P'_x$. All of those data structures use $O(N/B)$ blocks of space. All data structures C_j use at most $s(N)$ blocks of space.

The query is answered in the same way as in Lemma 1. Given a query $Q = [a, b] \times [c, +\infty)$, let $a' = succ(a, P'_x)$, $b' = pred(b, P'_x)$. If $a \leq b$, then all points in $[a', b] \times [c, +\infty)$ can be reported in constant time as describd in the proof of Lemma 1. All points in $[a, a) \times [c, +\infty)$ and all points in $(b, b] \times [c, +\infty)$ can be reported with two dominance reporting queries to interval data structures. Thus the query is answered in $2q_2(\log_B N) + O(1) + O(T/B)$ time. If $a > b$, then $[a, b]$ is contained in one interval, and all points can be reported with one three-sided query to an interval data structure in $q_1(\log_B N) + O(1) + O(T/B)$ time.

Lemma 3. *There exists a data structure that uses $O(\frac{N}{B} \log_B^* N)$ blocks of space and supports three-sided range reporting queries on $N \times \mathbb{N}$ grid in $O(\log_B^* N + T/B)$ time.*

Proof. Start with a data structure of [6] that uses $O(N/B)$ blocks and supports three-sided queries in $O(\log_B N)$ time and apply Lemma 2 $\log_B^* N$ times. The total number of recursive calls to interval data structures is $O(\log_B^* N)$.

Finally, we can construct a linear space data structure

Lemma 4. *There exists a data structure that uses $O(\frac{N}{B})$ blocks of space and supports three-sided range reporting queries on $N \times \mathbb{N}$ grid in $O(\log_B^{(k)} N + T/B)$ time for an arbitrary integer constant k.*

Proof. We start with the data structure of [6] and apply k times Lemma 2.

4 Range Reporting on a $U \times U$ Grid

Applying the reduction to rank space technique (see e.g. [10],[2]) and Statement 1 to Lemma 4 we obtain a $O(N/B)$ space data structure for range reporting on a $U \times \mathbb{N}$ grid

Lemma 5. *There exists a data structure that uses $O(\frac{N}{B})$ blocks of space and supports three-sided range reporting queries on $U \times \mathbb{N}$ grid in $O(\log_2 \log_B U + T/B)$ time.*

A $O(s(N))$ space data structure that supports three-sided range reporting queries can be transformed into a $O(s(N) \log_2 N)$ data structure that supports general orthogonal range reporting queries (see e.g. [21], [6]). For completeness, we describe this construction in the Appendix. Thus we obtain:

Theorem 1. *There exists a $O((N/B) \log_2 N)$ space data structure that supports two-dimensional range reporting queries on a $U \times U$ grid in $O(\log_2 \log_B U + T/B)$ time.*

It was shown in [15] that two-dimensional orthogonal emptiness queries have the same complexity as predecessor queries (see also [7]). Therefore, according to the lower bound of [19], the data structure of Theorem 1 achieves optimal query time. A data structure for three-sided queries with $o(\log_2 \log_B U + T/B)$ query time would imply a data structure for two-dimensional range reporting queries with $o(\log_2 \log_B U + T/B)$ query time. Hence, the query time of the data structure of Lemma 5 is also optimal.

Unlike many other external memory data structures, we assume in Theorem 1 and Lemma 5 that the set of allowed operations includes multiplications and divisions. If the set of operations is restricted by comparisons, additions, indirect addressing, and bit shift operations, we can apply Statement 2 instead of Statement 1 and achieve the following result:

Theorem 2. *There exists a data structure that uses $O(\frac{N}{B})$ blocks of space and supports three-sided range reporting queries on a $U \times \mathbb{N}$ grid in $O(\sqrt{\log_B U} + T/B)$ time. There exists a $O((N/B) \log_2 N)$ space data structure that supports two-dimensional range reporting queries on a $U \times U$ grid in $O(\sqrt{\log_B U} + T/B)$ time. The set of operations consists of comparisons, additions, indirect addressing, and bit shift operations.*

References

1. Aggarwal, A., Vitter, J.S.: The Input/Output Complexity of Sorting and Related Problems. Communications of the ACM 31(9), 1116–1127 (1988)
2. Alstrup, S., Brodal, G.S., Rauhe, T.: New Data Structures for Orthogonal Range Searching. In: Proc. FOCS, pp. 198–207 (2000)
3. Amir, A., Efrat, A., Indyk, P., Samet, H.: Efficient Regular Data Structures and Algorithms for Dilation, Location, and Proximity Problems. Algorithmica 30(2), 164–187 (2001)

4. Andersson, A.: Faster Deterministic Sorting and Searching in Linear Space. In: Proc. FOCS, pp. 135–141 (1996)
5. Arge, L.: External Memory Data Structures. In: Meyer auf der Heide, F. (ed.) ESA 2001. LNCS, vol. 2161, pp. 1–29. Springer, Heidelberg (2001)
6. Arge, L., Samoladas, V., Vitter, J.S.: On Two-Dimensional Indexability and Optimal Range Search Indexing. In: Proc. PODS, pp. 346–357 (1999)
7. Beame, P., Fich, F.E.: Optimal Bounds for the Predecessor Problem and Related Problems. J. Comput. Syst. Sci. 65, 38–72 (2002)
8. de Berg, M., van Kreveld, M.J., Snoeyink, J.: Two- and Three-Dimensional Point Location in Rectangular Subdivisions. J. Algorithms 18(2), 256–277 (1995)
9. Chan, T.M.: Point Location in o(log n) Time, Voronoi Diagrams in o(n log n) Time, and Other Transdichotomous Results in Computational Geometry. In: Proc. FOCS, pp. 333–344 (2006)
10. Chazelle, B.: A Functional Approach to Data Structures and its Use in Multidimensional Searching. SIAM J. on Computing 17, 427–462 (1988)
11. van Emde Boas, P., Kaas, R., Zijlstra, E.: Design and Implementation of an Efficient Priority Queue. Mathematical Systems Theory 10, 99–127 (1977)
12. Fries, O., Mehlhorn, K., Näher, S., Tsakalidis, A.K.: A log log n Data Structure for Three-Sided Range Queries. Information Processing Letters 25(4), 269–273 (1987)
13. Grossi, R., Italiano, G.F.: Efficient Splitting and Merging Algorithms for Order Decomposable Problems. Information and Computation 154, 1–33 (1999)
14. Icking, C., Klein, R., Ottmann, T.: Priority Search Trees in Secondary Memory (Extended Abstract). In: Göttler, H., Schneider, H.-J. (eds.) WG 1987. LNCS, vol. 314, pp. 84–93. Springer, Heidelberg (1988)
15. Miltersen, P.B., Nisan, N., Safra, S., Wigderson, A.: On Data Structures and Asymmetric Communication Complexity. J. Comput. Syst. Sci. 57, 37–49 (1998)
16. Nekrich, Y.: A Data Structure for Multi-Dimensional Range Reporting. In: Proc. SoCG 2007, pp. 344–353 (2007)
17. Overmars, M.H.: Efficient Data Structures for Range Searching on a Grid. J. Algorithms 9(2), 254–275 (1988)
18. Pătraşcu, M.: Planar Point Location in Sublogarithmic Time. In: Proc. FOCS, pp. 325–332 (2006)
19. Pătraşcu, M., Thorup, M.: Time-space Trade-offs for Predecessor Search. In: Proc. STOC, pp. 232–240 (2006)
20. Ramaswamy, S., Subramanian, S.: Path Caching: A Technique for Optimal External Searching. In: Proc. PODS, pp. 25–35 (1994)
21. Subramanian, S., Ramaswamy, S.: The P-range Tree: A New Data Structure for Range Searching in Secondary Memory. In: Proc. SODA, pp. 378–387 (1995)
22. Vitter, J.S.: External Memory Algorithms and Data Structures: Dealing with Massive Data. ACM Computing Surveys 33(2), 209–271 (2001)
23. Willard, D.E.: New Trie Data Structures Which Support Very Fast Search Operations. Journal of Computer and System Sciences 28(3), 379–394 (1984)

Appendix

In this Appendix we show how an $s(N)$ space data structure for three-sided range reporting queries can be transformed into a $O(s(N)\log_2 N)$ space data structure for two-dimensional range reporting queries using a standard technique (see e.g.

[21], or see [16] for a description of a variant of this technique for a bounded universe).

Using a reduction to rank space technique [10], [2] a two-dimensional range reporting query on a $U \times U$ grid can be reduced to a two-dimensional range reporting query on an $N \times N$ grid so that query time is increased by an additive factor of $O(\log_2 \log_B U)$. Now $P_x \subset [1, N]$, where P_x is the set of x-coordinates of all points in P. The interval $U^0 = [0, N-1]$ is divided into two intervals $U^1 = [0, N/2-1]$ and $U^2 = [N/2, N-1]$ of size $N/2$. Each interval U^i is recursively sub-divided into two intervals U^{2i} and U^{2i+1} of equal size. This division continues as long as the interval U^i contains elements from P_x and the size of the interval is bigger than 1. For every interval U^i there are two data structures for three-sided range reporting queries that contain all points whose x-coordinates belong to U^i and support three-sided queries that are open to the left and three-sided queries that are open to the right. Every such data structure uses linear space and supports queries in $O(\log_2 \log_B N + T/B)$ time. Hence all data structures for three-sided queries use $O(N \log_2 N)$ space. An interval U^i *splits* an interval $[a, b]$, if either a or b belongs to U^i, but $[a, b] \not\subset U^i$. The bounds of all non-empty intervals of size $N/2^l$ are stored in a data structure D^l that is implemented according to Statement 1. Using D^l, we can find for an arbitrary x a non-empty interval U^i of size $N/2^l$ that contains x, or report that no such U^i exists in $O(\log_2 \log_B N)$ time.

Suppose that a query $[a, b] \times [c, d]$ is to be answered. Using table look-up, we can find in $O(1)$ time such k that $N/2^k \leq b - a + 1 < N/2^{k-1}$. Then either for $m = k - 1$ or for $m = k$ or for $m = k + 1$, there are two consecutive intervals U^i, U^{i+1} of size $N/2^m$ that split $[a, b]$ (one of those intervals or both of them may be empty). Such intervals $U^i = [l^i, r^i]$ and $U^{i+1} = [r^i + 1, r^{i+1}]$ can be found in $O(\log_2 \log_B N)$ time with help of D^{k-1},D^k, and D^{k+1}. After this, a query $[a, b] \times [c, d]$ can be answered by answering two three-sided queries $[a, r^i + 1) \times [c, d]$ and $(r^i, b] \times [c, d]$ to data structures that store the points whose x-coordinates belong to U^i and U^{i+1} respectively.

Approximate Range Searching in External Memory

Micha Streppel[1,⋆] and Ke Yi[2,⋆⋆]

[1] NCIM-Groep, The Netherlands
m.streppel@ncim.nl
[2] Department of Computer Science and Engineering,
Hong Kong University of Science and Technology,
Clear Water Bay, Kowloon, Hong Kong
yike@cse.ust.hk

Abstract. In this paper, we present two linear-size external memory data structures for approximate range searching. Our first structure, the *BAR-B-tree*, stores a set of N points in $\mathbb{R}^d$ and can report all points inside a query range Q by accessing $O(\log_B N + \epsilon^\gamma + k_\epsilon/B)$ disk blocks, where B is the disk block size, $\gamma = 1 - d$ for convex queries and $\gamma = -d$ otherwise, and k_ϵ is the number of points lying within a distance of $\epsilon \cdot \mathrm{diam}(Q)$ to the query range Q. Our second structure, the *object-BAR-B-tree*, is able to store objects of arbitrary shapes of constant complexity and provides similar query guarantees. In addition, both structures also support other types of range searching queries such as range aggregation and nearest-neighbor. Finally, we present I/O-efficient algorithms to build these structures.

1 Introduction

Range searching is one of the most studied topics in computational geometry. In the basic problem, range reporting, we would like to build a data structure on a set of N points in $\mathbb{R}^d$ such that given a query range Q, all points inside Q can be reported efficiently. As the data sets are often massive in modern applications such as spatial databases, GIS, etc., the resulting data structures often have to be stored on disks. Thus, it is important to design efficient *external memory*, or *I/O-efficient* data structures that optimize disk block transfers instead of CPU time. The design of external memory range searching structures, sometimes called *spatial indexes*, has attracted a lot of interest in both the algorithm and database communities in the past decades.

Ideally, one would like the structure to have a linear size and logarithmic query cost. Unfortunately, this cannot be achieved except for very restricted query ranges. In two dimensions, when the queries are half-spaces, linear space

⋆ Work done while the author was at TU Eindhoven. Supported in part by the Netherlands Organisation for Scientific Research (N.W.O.) under project no. 612.065.203.
⋆⋆ Supported in part by Hong Kong Direct Allocation Grant (DAG07/08).

T. Tokuyama (Ed.): ISAAC 2007, LNCS 4835, pp. 536–548, 2007.

and $O(\log_B N + k/B)$ queries can be simultaneously achieved [1], where B is the disk block size, and k is the output size. However, if the queries are axis-parallel rectangles, in order to achieve a query bound of $O(\log_B N + k/B)$, a super-linear space of $\Omega(\frac{N}{B}\frac{\log(N/B)}{\log\log_B N})$ disk blocks is required, and there is also such a structure matching this bound [7]. If linear space is required, the best obtainable query bound is $O((N/B)^\epsilon + k/B)$ for any constant $\epsilon > 0$ [7]. These bounds become much higher in dimensions greater than 2. We refer the reader to the surveys [14,20,5] for other external memory range searching structures.

Given the theoretical hardness of the problem, practitioners often seek heuristic-based structures. Among them the R-tree and its many variants [16] tend to perform well. In addition, R-trees have several appealing features that make them a popular choice in practice. First, an R-tree usually has a small size, typically not much larger than the raw data size. Second, they support arbitrary query ranges, and can store not only points, but also objects of other shapes. Third, besides the basic range reporting query, they also support other kinds of range searching queries, for example nearest-neighbor, point location, aggregation queries, etc. Fourth, they easily generalize to higher dimensions. As a result, the R-trees have received tremendous research attention with many variants proposed, and are heavily used in practice, in spite of the their lack of good performance guarantees. In fact it was shown [3] that in the worst case, a query has to visit $\Omega((N/B)^{1-1/d} + k/B)$ blocks using *any* variant of R-trees built on N points in $\mathbb{R}^d$. This lower bound is reached by a recently developed R-tree variant [6], but the result holds only if both the queries and objects are axis-parallel hypercubes. If the objects are just points in $\mathbb{R}^d$, other practical structures such as the K-D-B tree [19], the quad tree [20], etc., are also often used.

Approximate range searching and the BAR-tree. Given the fact that exact range searching either uses non-linear storage or incurs super-logarithmic query time, it is natural to seek for approximate solutions. The concept of *ϵ-approximate range searching* was first introduced by Arya and Mount [8]. Here one considers, for a parameter $\epsilon > 0$ and a query range Q of constant complexity, the ϵ-extended query range Q_ϵ, which is the locus of points lying at distance at most $\epsilon \cdot \text{diam}(Q)$ from Q, where $\text{diam}(Q)$ is the diameter of Q. For a point set P of N points in $\mathbb{R}^d$, the following approximate range searching queries can be defined:

(Q1) *Range reporting:* Return a set P^* such that $P \cap Q \subseteq P^* \subseteq P \cap Q_\epsilon$.

(Q2) *Range aggregation:* Supposing each point $p \in P$ is associated with a weight $\omega(p) \in \mathbb{R}$, compute $\bigoplus_{p \in P^*} \omega(p)$ for some P^* such that $P \cap Q \subseteq P^* \subseteq P \cap Q_\epsilon$, where $\oplus$ is an associative and commutative operator. We say that the aggregation is *duplicate-insensitive* if $x \oplus x = x$ for any x. For example MAX is a duplicate-insensitive aggregation while + is not.

(Q3) *Nearest-neighbor:* For a query point q, return a point $p \in P$ such that $d(p, q) \leq (1 + \epsilon)d(p^*, q)$, where p^* is the true nearest neighbor of q and $d(p, q)$ is the Euclidean distance between p and q.

These problems were first considered by Arya and Mount [8], who proposed the BBD-tree. Later, a similar, but simpler structure, called the BAR-tree, was

proposed by Duncan et al. [13]. Our structures will be based on the BAR-tree, which we describe briefly below.

A BAR-tree [13,12] is a binary tree $\mathcal{T}$ that represents a binary space partition (BSP). We briefly describe the two-dimensional version below; generalization to $\mathbb{R}^d$ is similar. Each data point $p \in P$ is stored at a leaf of $\mathcal{T}$. Each node $u \in \mathcal{T}$ is associated with a region R_u that encloses all the points stored below u. The region associated with the root of $\mathcal{T}$ is the entire $\mathbb{R}^2$. For any internal node u, R_u is partitioned into two sub-regions R_v and R_w where v and w are the children of u. A (Q1) range reporting query Q can be answered using such a BSP by visiting all nodes of $\mathcal{T}$ recursively whose regions intersect Q. To support range aggregation queries, we in addition store at u the aggregate of the weights of all points stored below u. For a (Q2) query Q, we start from the root of $\mathcal{T}$ and traverse the tree while keeping a running aggregate. The only difference here is that we skip an entire subtree at some u if R_u is either completely inside Q_ϵ or outside Q. A (Q3) query can be answered by keeping a priority queue storing all the candidate nodes [12].

In a BAR-tree, all the regions R_u are convex, have aspect ratios bounded by some constant, and the boundary of each R_u consists of a constant number of vertical, horizontal, and diagonal line segments. Duncan et al. [13,12] proved that even under these constraints, using at most two splits, any R_u can be partitioned into 2 or 3 cells, such that the number of points in any cell is at most a constant fraction of the number of points in R_u. Thus the height of the tree can be bounded by $O(\log N)$. Note however that some subtrees in a BAR-tree may not be balanced, since sometimes the first split may have to partition the points in R_u into two subsets with drastically different cardinalities.

A BAR-tree obviously uses linear space, and because of the properties of the regions, the number of nodes visited during a query can be effectively bounded using a packing argument [8]. As a result, it is shown [13] that (Q1) takes $O(\log N + \epsilon^\gamma + k_\epsilon)$ time for any query range Q, where $\gamma = 1 - d$ for convex ranges and $\gamma = -d$ otherwise, and k_ϵ is the number of points inside Q_ϵ.[1] (Q2) can be answered in time $O(\log N + \epsilon^\gamma)$ and (Q3) in time $O(\log N + \epsilon^{1-d} \log(1/\epsilon))$.

The I/O-model and previous work. For the analysis of external memory data structures, the standard *I/O model* by Aggarwal and Vitter [4] is often used. In this model, the memory has a limited size M but any computation in memory is free. In one I/O a disk block consisting of B items are read from or written to the external memory. Only the number of I/Os is considered when analyzing the cost of an algorithm. The size of a data structure is measured in the number of disk blocks it occupies. Many fundamental problems have been solved in the I/O model. For example, sorting N elements takes $\text{sort}(N) = \Theta(N/B \log_{M/B}(N/B))$ I/Os. Please refer to [21,5] for comprehensive surveys on I/O-efficient algorithms and data structures.

[1] As noted by Haverkort et al. [15], the actual bound of (Q1) is $O(\log N + \min_\epsilon\{\epsilon^\gamma + k_\epsilon\})$ since Q_ϵ is only used in the analysis and not by the query algorithm, which just uses Q to visit $\mathcal{T}$ and always reports the correct answers $P \cap Q$.

Table 1. Summary of our results for the BAR-B-tree on a set of N points and the object-BAR-B-tree on a set of N objects in $\mathbb{R}^d$ for any fixed d. For (Q1) and (Q2), $\gamma = 1 - d$ if the range is convex, and $-d$ otherwise. The update bound is amortized. $^{(*)}$This bound holds only for duplicate-insensitive aggregations.

	BAR-B-tree	object-BAR-B-tree
Size	$O(N/B)$	$O(\lambda N/B)$
Construction	$O(\mathrm{sort}(N))$	$O(\mathrm{sort}(\lambda N))$
(Q1)	$O(\log_B N + \epsilon^\gamma + k_\epsilon/B)$	$O(\log_B N + \lceil\lambda/B\rceil\epsilon^\gamma + \lambda k_\epsilon/B)$
(Q2)	$O(\log_B N + \epsilon^\gamma)$	$O(\log_B N + \lceil\lambda/B\rceil\epsilon^\gamma)^{(*)}$
(Q3)	$O(\log_B N + \epsilon^{1-d}(1 + \frac{1}{B}\log_{M/B}(1/\epsilon)))$	$O(\log_B N + \lceil\lambda/B\rceil\epsilon^{1-d}(1 + \frac{1}{B}\log_{M/B}(1/\epsilon)))$
Update	$O(\log_B N + \frac{1}{B}\log_{M/B}(N/B)\log(N/B))$	-

There has been some work on efficient disk layouts of the BAR-tree. By using standard techniques such as a breadth-first blocking scheme and I/O-efficient priority queues, it is pretty straightforward to lay out a BAR-tree on disk such that (Q2) and (Q3) can be answered with $O(\log_B N + \epsilon^\gamma)$ and $O(\log_B N + \epsilon^{1-d}(1 + \frac{1}{B}\log_{M/B}(1/\epsilon)))$ I/Os, respectively. For a range reporting query (Q1) that has a potentially large output, it is crucial to have an output term of $O(k/B)$ rather than $O(k)$. As typical values of B are on the order of hundreds to thousands, the difference between $O(k/B)$ I/Os and $O(k)$ I/Os can be significant.

In his thesis [12] Duncan gave an I/O-efficient variant of the BAR-tree, which uses a breadth-first blocking scheme. The number of blocks visited for answering a (Q1) query is claimed to be $O(\log_B N + \epsilon^\gamma + k_\epsilon/B)$. However this result relies on the incorrect premise that all blocks contain $\Theta(B)$ nodes. Some leaves may contain a small number of points and the query bound is in fact $O(\log_B N + \epsilon^\gamma + k_\epsilon)$ in the worst case. Agarwal et al. [2] gave a general framework for externalizing and dynamizing *weight-balanced partitioning trees* such as the BAR-tree. Like Duncan [12], they use a breadth-first blocking scheme for storing the BAR-tree on disk. To remove the assumption made by Duncan they group blocks together which contain too few nodes. As a result there is at most one block containing too few nodes. This improvement ensures that the resulted layout only uses $O(N/B)$ disk blocks, but (Q1) query cost is still $O(\log_B N + \epsilon^\gamma + k_\epsilon)$, since the k_ϵ points that need to be visited could spread to $\Omega(k_\epsilon)$ blocks.

Our results. We obtain two main results in this paper. We first give a new blocking scheme for the BAR-tree that yields the first disk-based data structure, the *BAR-B-tree*, which answers all of the aforementioned approximate queries efficiently. In particular, the BAR-B-tree answers an approximate range reporting query (Q1) in the desired $O(\log_B N + \epsilon^\gamma + k_\epsilon/B)$ I/Os[2] , i.e., achieving an $O(\log_B N)$ search term and an $O(k_\epsilon/B)$ output term simultaneously. Such terms are optimal when external memory structures are concerned [5]. Unfortunately it seems difficult to reduce the $O(\epsilon^\gamma)$ term. The bounds for other queries and

[2] By the same observation of [15], the actual bound is $O(\log_B N + \min_\epsilon\{\epsilon^\gamma + k_\epsilon/B\})$, but we will not write out $\min_\epsilon$ explicitly.

operations match the previous results on externalizing BAR-trees [12,2]. In addition, we can also construct and update the BAR-B-tree efficiently. Please see Table 1 for the detailed results.

Next, we generalize the BAR-B-tree to the *object-BAR-B-tree*, which stores not just points, but arbitrary spatial objects of constant complexity. The approximate range searching queries (Q1), (Q2), and (Q3) are generalized to objects as follows. Let S be a set of objects in $\mathbb{R}^d$ and Q the query range. For (Q1) and (Q2), we return a subset $S^* \subseteq S$ (or the aggregation $\bigoplus_{o \in S^*} \omega(o)$) where S^* includes all objects in S that intersect Q, does not include any object that does not intersect Q_ϵ, and may optionally include some objects that intersect Q_ϵ but not Q. For (Q3), the definition remains the same with the distance definition between an object o and the query point q being $d(o, q) = \min_{p \in o} d(p, q)$. Our idea is based on range searching data structures for *low-density scenes* [11,10]. The *density* of S is the smallest number λ such that the following holds: any ball b is intersected by at most λ objects $o \in S$ with $\rho(o) \geq \rho(b)$ where $\rho(o)$ denotes the radius of the smallest enclosing ball of o [10]. It is believed that for many realistic inputs, λ is small. For example, if all objects of S are disjoint and *fat* (i.e., have bounded aspect ratio), then λ is a constant. The object-BAR-B-tree exhibits the same performance bounds as the BAR-B-tree if λ is a constant, and the costs grow roughly linearly with λ for all operations except for updates. Please refer to Table 1 for the detailed bounds of various operations.

To summarize, with the BAR-B-tree and the object-BAR-B-tree, we present the first external memory data structures that have all the nice features the R-trees have, and in addition provide provable guarantees, albeit in the approximate sense. In addition, these two structures are not difficult to implement, and we expect them to be fairly practical as well. It would be interesting to compare them with R-trees, K-D-B trees, etc., to see how well they behave in practice.

2 The BAR-B-tree

In this section we describe the BAR-B-tree, an efficient layout for the BAR-tree on disk that achieves all the desired bounds listed in Table 1. We introduce our two-stage blocking scheme in Section 2.1, and analyze its query cost when answering a range searching query (Q1) in Section 2.2. The analysis for (Q2) and (Q3) is similar to that in [2,12], and hence is omitted. Finally we briefly talk about construction and updates in Section 2.3. For the remainder of the paper we assume that $\mathcal{T}$ has at least B nodes, otherwise the problem is trivial.

2.1 The Blocking Scheme

For any node $u \in \mathcal{T}$, let $\mathcal{T}_u$ be the subtree rooted at u, and we define $|\mathcal{T}_u|$, the size of $\mathcal{T}_u$, to be the number of nodes in $\mathcal{T}_u$ (including u). Our blocking scheme consists of two stages. In the first stage the tree is blocked such that for any $u \in \mathcal{T}$, $\mathcal{T}_u$ is stored in $O(\lceil |\mathcal{T}_u|/B \rceil)$ blocks. As we will see, this property will guarantee the $O(k_\epsilon/B)$ term in the query bound. However, a root-to-leaf path

in $\mathcal{T}$ may be covered by $\Theta(\log N)$ such blocks. In the second stage we make sure that any root-to-leaf path can be traversed by accessing $O(\log_B N)$ blocks, leading to the desired bound.

Algorithm 1. Algorithm to construct tree-blocks

Input: a binary tree $\mathcal{T}$
Output: a set of tree-blocks stored on disk

```
1  initialize S := {root of T}, and a block B := ∅;
2  while S ≠ ∅ do
3      remove any node u from S;
4      initialize a queue Q := {u};
5      while Q ≠ ∅ do
6          remove the first node v from Q, let v1, v2 be v's children;
7          if |T_v| ≤ B then
8              put T_v in a new block B';
9              write B' to disk;
10         else if |T_v1| ≥ B/2 and |T_v2| ≥ B/2 then
11             add v to B;
12             add v1, v2 to Q;
13         else
14             suppose |T_v1| < B/2;
15             if |B| + |T_v1| + 1 ≤ B then
16                 add v and T_v1 to B;
17                 add v2 to Q;
18             else
19                 add v to S;
20         if |B| = B then
21             write B to disk and reset B := ∅;
22             add all nodes of Q to S;
23             set Q := ∅;
24     if |B| ≠ ∅ then
25         write B to disk and reset B := ∅;
```

First stage. In the first stage we block the tree into *tree-blocks* that satisfy the property mentioned above. The blocking procedure is detailed in Algorithm 1. We traverse the tree $\mathcal{T}$ in a top-down fashion, and keep in a set $\mathcal{S}$ all nodes u for which a block will be allocated such that u is the topmost node in the block. Initially $\mathcal{S}$ only contains the root of $\mathcal{T}$. For any node $u \in \mathcal{S}$, we find a connected subtree rooted at u to fit in one block using an adapted breadth-first strategy with a queue $\mathcal{Q}$. Throughout the blocking algorithm we maintain the invariant that $|\mathcal{T}_u| \geq B/2$ for any u that is ever added to $\mathcal{S}$ or $\mathcal{Q}$. The invariant is certainly true when the algorithm initializes (line 1).

For a node $u \in \mathcal{S}$, we fill a block with a top portion of $\mathcal{T}_u$ by an adapted breadth-first search (line 4–23). The BFS starts with $\mathcal{Q} = \{u\}$ (line 4), which is consistent with the invariant since u is a node from $\mathcal{S}$. For each node v encountered in the BFS search, we distinguish among the following three cases. (a) If $|\mathcal{T}_v| \leq B$, then we allocate a new block to store the entire $\mathcal{T}_v$ (line 7–9). Note that this block contains at least $B/2$ nodes by the invariant. (b) Let v_1, v_2 be the two children of v. If both $\mathcal{T}_{v_1}$ and $\mathcal{T}_{v_2}$ have more than $B/2$ nodes, then we add v to the block and continue the BFS process (line 10–12). It is safe to add v_1, v_2 to $\mathcal{Q}$ as we have ensured the invariant. (c) Otherwise, it must be the case that one of the subtrees is smaller than $B/2$ nodes while the other one has more than $B/2$ nodes. Without loss of generality we assume $|\mathcal{T}_{v_1}| < B/2$, and then check if $\mathcal{T}_{v_1}$ plus v itself still fits in the current block. If so we put v and the entire $\mathcal{T}_{v_1}$ in the current block, add v_2 to $\mathcal{Q}$ and continue

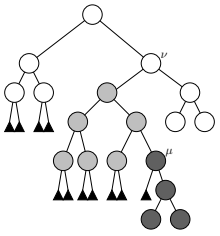


Fig. 1. Three tree-blocks (white, light gray and dark gray) obtained using the blocking scheme for $B = 8$. A black triangles denotes a subtree of size at least $B/2$. The right subtree of ν is placed completely in the white block. The node μ and its right subtree do not fit in the light gray block so a new block must be started at μ.

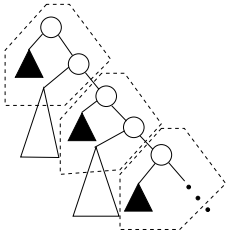


Fig. 2. A bad example for tree-blocks. All black triangles represent subtrees of size $B/2-1$, and all white triangles represent subtrees of sufficiently large size. The rightmost path L will be split into $\Theta(\log N)$ blocks.

the BFS (notice that $|\mathcal{T}_{v_2}| > B/2$); else we put v into $\mathcal{S}$, and will allocate a new block for v (line 14–19). Notice that the second time v is considered by the algorithm, line 19 will never be reached again since with a new empty block, $\mathcal{T}_{v_1}$ and v must be able to fit. Please refer to Figure 1 for an illustration of this blocking algorithm.

Lemma 1. *For any $u \in \mathcal{T}$, the nodes in $\mathcal{T}_u$ are stored in $O(\lceil |\mathcal{T}_u|/B \rceil)$ blocks.*

Proof. First consider the case where u is the topmost node in some block, i.e., u has been added to $\mathcal{S}$. Suppose a tree-block $\mathcal{B}$ contains less than $B/2$ nodes of the tree $\mathcal{T}_u$. There is at least one block below $\mathcal{B}$ since by the invariant the subtree of $\mathcal{T}$ rooted at the topmost node in $\mathcal{B}$ has size at least $B/2$. We claim that at least one block $\mathcal{B}'$ directly below $\mathcal{B}$ contains at least $B/2$ nodes. Now suppose for a contradiction that no block directly below $\mathcal{B}$ contains more than $B/2$ nodes. Let $\mathcal{B}'$ be a block below $\mathcal{B}$ containing less than $B/2$ nodes and let v be the highest node in $\mathcal{B}'$. The node v was not placed in $\mathcal{B}$ either because the size of $\mathcal{T}_v$ is between $B/2$ and B (case (a)) or because the size of one of its subtrees, say $\mathcal{T}_{v_1}$, is less than $B/2$ (case(c)). The former case immediately leads to a contradiction. The latter case also leads to a contradiction since $\mathcal{T}_{v_1}$, together with v, fits in $\mathcal{B}$ and would then have been placed in $\mathcal{B}$. So there must be a block $\mathcal{B}'$ below $\mathcal{B}$ whose size is at least $B/2$. We charge $\mathcal{B}$ to $\mathcal{B}'$. Each block containing at least $B/2$ nodes is charged at most once, namely by the block directly above it. The number of tree-blocks is thus $O(\lceil |\mathcal{T}_u|/B \rceil)$.

Next consider the case where $u \in \mathcal{B}$ but u is not the topmost node in $\mathcal{B}$. Let $u_1, \ldots, u_t$ be the t nodes of $\mathcal{T}_u$ stored immediately below $\mathcal{B}$. By the blocking algorithm's invariant, we have $|\mathcal{T}_{u_i}| \geq B/2$, so $|\mathcal{T}_u| > t \cdot B/2$, or $t < 2|\mathcal{T}_u|/B$. Applying the case above, the number of blocks used to store $\mathcal{T}_u$ is thus $1 + O(\sum_{i=1}^{t}(\lceil |\mathcal{T}_{u_i}|/B \rceil) = O(|\mathcal{T}_u|/B + t + 1) = O(\lceil |\mathcal{T}_u|/B \rceil)$.

The blocked BAR-tree resulting after the first stage might have depth as bad as $\Theta(\log N)$, as illustrated by the following example. Consider a root-to-leaf path L in a BAR-tree $\mathcal{T}$ of length $\Theta(\log N)$. In a BAR-tree, at every other node on L the split may be unbalanced. In particular each unbalanced split at a node v might have one subtree, say $\mathcal{T}_{v_1}$, of size $B/2 - 1$ (see Figure 2). Therefore our algorithm will try to store v and $\mathcal{T}_{v_1}$ in the current block. However this attempt will always fail, and we will end up with storing every two nodes on L in a different block, thus L is split over $\Theta(\log N)$ blocks. In the second stage we introduce *path-blocks*, which ensure that $O(\log_B N)$ blocks have to be accessed in order to visit all nodes on any root-to-leaf path.

Second stage. To identify the places where a path-block has to be introduced, we visit $\mathcal{T}$ in a top-down fashion. For a node u, if $|\mathcal{T}_u| \leq B$ we stop. From Lemma 1 we know that $\mathcal{T}_u$ is already covered by $O(1)$ tree-blocks. Otherwise we consider the top subtree of B nodes of $\mathcal{T}_u$ obtained by a BFS starting from u. We denote this subtree by $\widehat{\mathcal{T}}_u$. We check all root-to-leaf paths in $\widehat{\mathcal{T}}_u$. If there is at least one such path that is covered by more than c tree-blocks for some integer constant $c \geq 2$, then we introduce a path-block that stores $\widehat{\mathcal{T}}_u$. We also remove all nodes of $\widehat{\mathcal{T}}_u$ from the tree-blocks where they are stored. Finally we continue this process recursively with each subtree below $\widehat{\mathcal{T}}_u$.

This completes our two-stage blocking scheme. With the introduction of path-blocks, now we have the following.

Lemma 2. *Any root-to-leaf path in $\mathcal{T}$ can be traversed by accessing $O(\log_B N)$ blocks.*

Proof. Note that for any root-to-leaf path L in any $\widehat{\mathcal{T}}_u$, if it does not reach a leaf of $\mathcal{T}$, then L is at least $\log B$ long. Therefore, we can traverse $\log B$ consecutive nodes in any root-to-leaf path of $\mathcal{T}$ by accessing $O(1)$ blocks, hence the proof.

Since any path-block has at least $B/2$ nodes, it is easy to see that Lemma 1 still holds. In particular, we obtain the desired space bound for the BAR-B-tree.

Theorem 1. *A BAR-B-tree on N points in $\mathbb{R}^d$ takes $O(N/B)$ disk blocks.*

Since our blocking scheme has no redundancy, i.e., each node of $\mathcal{T}$ is stored in only one block, after the two-stage blocking process we can group blocks together such that all of them are at least half-full. So the space utilization of the BAR-B-tree can be at least 50%.

2.2 Analysis of the Range Reporting Query Time

Since no node is stored in multiple blocks we can use the standard query algorithm for BSPs, that is, we start from the root, and visit all nodes u of $\mathcal{T}$ where the region associated with u intersects with the query range Q. The traversal can be performed in either a BFS or DFS manner, with the use of an I/O-efficient stack or queue such that the extra overhead is $O(1)$ I/Os per B nodes. So we only need to bound the number of blocks that store all the visited nodes.

Theorem 2. *A (Q1) range reporting query Q in a BAR-B-tree can be answered by accessing $O(\log_B N + \epsilon^\gamma + k_\epsilon/B)$ blocks.*

Proof. Fix any ϵ. Note that any visited node must be of one of the following two types: (a) the nodes whose regions intersect Q and also the boundary of Q_ϵ, and (b) the nodes whose regions are completely contained in Q_ϵ. Note that some type-(b) nodes may not be visited by the query algorithm.

We start by proving a bound on the number of blocks containing the type-(a) nodes. From [13,12], we know that the number of such nodes is $O(\log N + \epsilon^\gamma)$. The $O(\log N)$ term comes from a constant number of root-to-leaf paths in $\mathcal{T}$. By Lemma 2 these nodes are covered by $O(\log_B N)$ blocks. So in total we need to access $O(\log_B N + \epsilon^\gamma)$ blocks for nodes of type (a).

Next we give a bound on the number of blocks that cover all the type-(b) nodes. These nodes are organized in t disjoint subtrees $\mathcal{T}_{u_1}, \ldots, \mathcal{T}_{u_t}$, such that $R_{u_i} \subseteq Q_\epsilon$ and $R_{p(u_i)} \not\subseteq Q_\epsilon$, where $p(u_i)$ denotes the parent of u_i. Note that since R_{u_i} is contained in Q_ϵ, $R_{p(u_i)}$ must intersect the boundary of Q_ϵ, i.e., a type-(a) node. Each parent $p(u_i)$ has only one child whose region is inside Q_ϵ, since otherwise $R_{p(u_i)}$ would be completely inside Q_ϵ. From [13,12] we know that there are in total $O(\epsilon^\gamma)$ type-(a) nodes who have a child associated with a region completely inside Q_ϵ, hence $t = O(\epsilon^\gamma)$.

Note that the subtree $\mathcal{T}_{u_i}$ stores at least $|\mathcal{T}_{u_i}|/2$ points of P inside Q_ϵ. By Lemma 1, $\mathcal{T}_{u_i}$ is stored in $O(\lceil |\mathcal{T}_{u_i}|/B \rceil)$ blocks. Thus the total number of blocks covering all the t subtrees is $O\left(\sum_{i=1}^{t} \lceil |\mathcal{T}_{u_i}|/B \rceil\right) = O\left(t + \sum_{i=1}^{t} |\mathcal{T}_{u_i}|/B\right) = O(\epsilon^\gamma + k_\epsilon/B)$.

2.3 Construction and Updates

The construction algorithm of the BAR-B-tree can be based on the "grid" technique [2] and uses $O(\text{sort}(N))$ I/Os. Due to space limit we omit the details from this abstract. We can also use the *partial rebuilding* technique [17,2] to handle insertions and deletions for the BAR-B-tree. To insert a point into the BAR-B-tree, we first follow a root-to-leaf path to find the leaf block where the point should be located. According to Lemma 2 this takes $O(\log_B N)$ I/Os. After inserting the point we check the nodes on this path to see if any of them contains too many points. Among all those nodes, we rebuild the whole subtree rooted at the highest one. Deletions can be handled similarly. Using standard analysis, it can be shown that the amortized cost of an update is $O(\log_B N + \frac{1}{B} \log_{M/B}(N/B) \log(N/B))$ I/Os, and we omit the details.

3 Extension to Objects: the Object-BAR-B-tree

In this section we show how to externalize the *object-BAR-tree* [11], for a set S of objects of constant complexity with density λ. We first briefly review the object-BAR-tree below.

The object-BAR-tree is based on the idea of *guarding sets* [9]. For a subset $X \subseteq S$, a set of points G_X is called a λ-*guarding set* (simply called a *guarding set* in the following) of X if the region associated with any leaf in the BAR-tree constructed on G_X intersects at most $O(\lambda)$ objects of X. To build the object-BAR-tree on S, we first for each object $o \in S$ compute a constant number of points, called the *guards* of o, with the property that the guards of any subset X of S form a guarding set for X. Let G be the set of all guards. We build the object-BAR-tree by first constructing a BAR-tree $\mathcal{T}$ on G, with the adaptation that whenever we are going to build a subtree for a region R with a subset $G' \subset G$, we delete all guards from G' whose objects do not intersect R. Then we store at each leaf of $\mathcal{T}$ all objects of S that intersect the region associated with the leaf. Because G is a guarding set of S, each leaf stores $O(\lambda)$ objects. It was shown [11] that a (Q1) query can be answered in time $O(\log N + \lambda(\epsilon^{\gamma} + k_{\epsilon}))$ using the object-BAR-tree.

3.1 Building the Object-BAR-B-tree

We first build all the guards with a scan over S. For $\mathbb{R}^2$ we can use the simple construction (Figure 3) of De Berg et al. [11]. For $\mathbb{R}^d$, $d \geq 3$ the construction is more involved and the details can be found in [18].

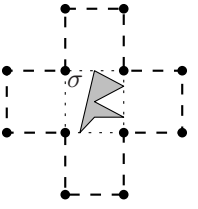


Fig. 3. The guards for an object in $\mathbb{R}^2$

Next we build the BAR-B-tree on the set of all guards G. The adaptation of removing guards during the construction as described above can easily be accommodated in the algorithm, and we can build and lay out the tree $\mathcal{T}$ on disk in $O(\text{sort}(\lambda N))$ I/Os. During the process we can also compute for each leaf v of $\mathcal{T}$, the set of at most $O(\lambda)$ objects that intersect the region R_v. We omit the technical details.

Finally, for each leaf block $\mathcal{B}$ of $\mathcal{T}$, we store all the intersecting objects consecutively on disk. More precisely, consider a block $\mathcal{B}$ and let L be the set of leaves stored in $\mathcal{B}$. The objects intersecting the regions of the nodes in L are stored together in one list as follows. Let $v_1, \cdots, v_{|L|}$ be the leaves in L ordered according to an in-order traversal of $\mathcal{T}$. We first store the objects intersecting R_{v_1}, then the objects intersecting R_{v_2}, etc. Note that an object might be stored more than once in the list. At every leaf v_i we store a pointer to the first and last object in the list intersecting R_{v_i}. Since each leaf has $O(\lambda)$ intersecting objects, each such list occupies $O(\lambda B/B) = O(\lambda)$ disk blocks, so the overall space usage of these lists is $O(\lambda N/B)$ blocks. This completes the description of the object-BAR-B-tree. Note that the object-BAR-B-tree automatically reduces to the BAR-B-tree when all the objects are points.

Theorem 3. *Let S be a set of N objects in $\mathbb{R}^d$ with density λ. An object-BAR-B-tree on S takes $O(\lambda N/B)$ blocks and can be constructed in $O(\text{sort}(\lambda N))$ I/Os.*

3.2 Analysis of the Query Cost

In this section we prove the bounds stated in Table 1 for the object-BAR-B-tree. The query bounds for (Q2) and (Q3) follow from the bounds on (Q2) and (Q3)

for the BAR-B-tree and the fact that for every visited leaf v, we now have to check the $\lceil \lambda/B \rceil$ blocks containing the objects intersecting R_v. However, note that since in an object-BAR-B-tree an object might be stored at several leaves, we can only handle duplicate-insensitive aggregations for (Q2). We are left with proving the query bound on ($Q1$).

Theorem 4. *Let S be a set of N objects of constant complexity in $\mathbb{R}^d$ with density λ. An object-BAR-B-tree for S answers a (Q1) query Q using $O(\log_B N + \lceil \lambda/B \rceil \epsilon^\gamma + \lambda k_\epsilon/B)$ I/Os, where k_ϵ is the number of objects intersecting Q_ϵ.*

Proof. The query cost of answering a range searching query Q consists of two parts: the cost to visit the nodes of $\mathcal{T}$ and the cost to read the object lists. Since $\mathcal{T}$ is a BAR-B-tree with possible removal of guards during construction, which only reduces the number of nodes, the cost of visiting $\mathcal{T}$ can still be bounded by Theorem 2. So we only concentrate on the cost of reading the object lists of the visited leaves.

Fix any ϵ. Any visited leaf must fall into one of the following two categories: either its region intersects Q and also the boundary of Q_ϵ, or its region is completely contained in Q_ϵ. There are at most ϵ^γ leaves of the former type [12]. For these leaves we can check all objects intersecting their regions using $O(\lceil \lambda/B \rceil)$ I/Os each.

The latter type of leaves can be covered in t disjoint subtrees $\mathcal{T}_{u_1}, \ldots, \mathcal{T}_{u_t}$, such that $R_{u_i} \subseteq Q_\epsilon$ and $R_{p(u_i)} \not\subseteq Q_\epsilon$, where $p(u_i)$ denotes the parent of u_i. Note that there are $O(\epsilon^\gamma)$ such subtrees [12,13]. For any u_i, let $k(u_i)$ denote the number of objects that intersect R_{u_i} and have at least one guard in R_{u_i}. Since $\mathcal{T}_{u_i}$ is a BAR-tree built on the $O(k(u_i))$ guards of these objects (with pruning), and each object has a guard in at least one of the leaves of $\mathcal{T}_{u_i}$, we have $|\mathcal{T}_{u_i}| = O(k(u_i))$. Furthermore, since each object intersecting Q_ϵ has guards in at most a constant number of these subtrees, we have $\sum_{i=1}^{t} k(u_i) = O(k_\epsilon)$.

Consider some $\mathcal{T}_{u_i}$, and let $v_1, v_2, \ldots$ be the leaves of $\mathcal{T}_{u_i}$ ordered according to an in-order traversal of $\mathcal{T}$. From our blocking algorithm for the BAR-B-tree, we know that these leaves are partitioned into $O(\lceil |\mathcal{T}_{u_i}|/B \rceil)$ pieces, each stored in a block. Since in a block, the objects intersecting consecutive leaves are also stored consecutively in the object list, the total number of I/Os to read these objects is $O(\lceil |\mathcal{T}_{u_i}|/B \rceil + \lambda |\mathcal{T}_{u_i}|/B) = O(\lceil \lambda |\mathcal{T}_{u_i}|/B \rceil)$. Thus, the total number of I/Os for reading the object lists for all the leaves whose regions are completely inside Q_ϵ is

$$O\left(\sum_{i=1}^{t} \left\lceil \frac{\lambda |\mathcal{T}_{u_i}|}{B} \right\rceil\right) = O\left(t + \sum_{i=1}^{t} \frac{\lambda |\mathcal{T}_{u_i}|}{B}\right) = O\left(t + \sum_{i=1}^{t} \frac{\lambda k(u_i)}{B}\right) = O(\epsilon^\gamma + \lambda k_\epsilon/B).$$

Updating the object-BAR-B-tree. The object-BAR-B-tree can be updated by first updating the BAR-B-tree $\mathcal{T}$, followed by updating the object lists. Since each object only has a constant number of guards, the cost of the former is the same as the update cost of the BAR-B-tree asymptotically. However, we do not have a

worst-case or amortized bound for the latter, as one object may intersect many regions of the leaves of $\mathcal{T}$. Nevertheless, since an object only intersects $O(\lambda)$ such regions on average over all stored objects, we expect the actual update cost to be small in practice.

References

1. Agarwal, P.K., Arge, L., Erickson, J., Franciosa, P., Vitter, J.: Efficient searching with linear constraints. Journal of Computer and System Sciences 61(2), 194–216 (2000)
2. Agarwal, P.K., Arge, L., Procopiuc, O., Vitter, J.S.: A framework for index bulk loading and dynamization. In: Proc. International Colloquium on Automata, Languages, and Programming, pp. 115–127 (2001)
3. Agarwal, P.K., de Berg, M., Gudmundsson, J., Hammar, M., Haverkort, H.J.: Box-trees and R-trees with near-optimal query time. Discrete and Computational Geometry 28(3), 291–312 (2002)
4. Aggarwal, A., Vitter, J.S.: The input/output complexity of sorting and related problems. Communications of the ACM 31(9), 1116–1127 (1988)
5. Arge, L.: External memory data structures. In: Abello, J., Pardalos, P.M., Resende, M.G.C. (eds.) Handbook of Massive Data Sets, pp. 313–358. Kluwer Academic Publishers, Dordrecht (2002)
6. Arge, L., de Berg, M., Haverkort, H.J., Yi, K.: The priority R-tree: A practically efficient and worst-case optimal R-tree. In: Proc. SIGMOD International Conference on Management of Data, pp. 347–358 (2004)
7. Arge, L., Samoladas, V., Vitter, J.S.: On two-dimensional indexability and optimal range search indexing. In: Proc. ACM Symposium on Principles of Database Systems, pp. 346–357 (1999)
8. Arya, S., Mount, D.M.: Approximate range searching. In: Proc. 11th Annual ACM Symposium on Computational Geometry, pp. 172–181 (1995)
9. de Berg, M., David, H., Katz, M.J., Overmars, M., van der Stappen, A.F., Vleugels, J.: Guarding scenes againt invasive hypercubes. Computational Geometry: Theory and Applications 26, 99–117 (2003)
10. de Berg, M., Katz, M.J., van der Stappen, A.F., Vleugels, J.: Realistic input models for geometric algorithms. In: Proc. 13th Annual ACM Symposium Computational Geometry, pp. 294–303 (1997)
11. de Berg, M., Streppel, M.: Approximate range searching using binary space partitions. Computational Geometry: Theory and Applications 33(3), 139–151 (2006)
12. Duncan, C.: Balanced Aspect Ratio Trees. PhD thesis, John Hopkins University (1999)
13. Duncan, C., Goodrich, M., Kobourov, S.: Balanced aspect ratio trees: Combining the advantages of k-d trees and octrees. Journal of Algorithms 38, 303–333 (2001)
14. Gaede, V., Günther, O.: Multidimensional access methods. ACM Computing Surveys 30(2), 170–231 (1998)
15. Haverkort, H.J., de Berg, M., Gudmundsson, J.: Box-trees for collision checking in industrial installations. In: Proc. ACM Symposium on Computational Geometry, pp. 53–62 (2002)
16. Manolopoulos, Y., Nanopoulos, A., Papadopoulos, A.N., Theodoridis, Y.: R-trees: Theory and Applications. Springer, Heidelberg (2005)

17. Overmars, M.H.: The Design of Dynamic Data Structures. LNCS, vol. 156. Springer, Heidelberg (1983)
18. Streppel, M.: Multifunctional Geometric Data Structures. PhD thesis, Faculty of Mathematics and Computer Science, TU Eindhoven (2007)
19. Robinson, J.: The K-D-B tree: A search structure for large multidimensional dynamic indexes. In: Proc. SIGMOD International Conference on Management of Data, pp. 10–18 (1981)
20. Samet, H.: Spatial data structures. In: Kim, W. (ed.) Modern Database Systems, The Object Model, Interoperability and Beyond, pp. 361–385. ACM Press and Addison-Wesley (1995)
21. Vitter, J.S.: External memory algorithms and data structures: Dealing with MASSIVE data. ACM Computing Surveys 33(2), 209–271 (2001)

Faster Treasure Hunt and Better Strongly Universal Exploration Sequences

Qin Xin*

Department of Informatics, The University of Bergen, Bergen, Norway and
Simula Research Lab, Oslo, Norway
Qin.Xin@ii.uib.no

Abstract. We study the explicit deterministic treasure hunt problem in an n-vertex network. This problem was firstly introduced by Ta-Shma, and Zwick in [9] [SODA'07]. It is the variant of the well known rendezvous problem in which one of the robot (the treasure) is always stationary. We obtain an $O(n^{c(1+\frac{1}{\lambda})})$-time solution for this problem, which significantly improves the currently best known result of running time $O(n^{2c})$ in [9], where c is a fixed constant from the construction of an universal exploration sequence in [8,9], λ is a constant integer and $\lambda \gg 1$. The treasure hunt problem motivates the study of strongly universal exploration sequences. We give a better explicit construction of strongly universal exploration sequences than the one in [9].

Keywords: design and analysis of algorithms, distributed computing, networks, rendezvous, strongly universal exploration sequences.

1 Introduction

In the *rendezvous problem* ([3,6,9]), two robots are placed in an unknown environment modeled by a finite, connected, undirected graph $G = (V, E)$. We assume that $|V| = n$. The size of the network, i.e., the number of vertices in the graph is not known to the robots. The edges incident on a vertex $u \in V$ are numbered $0, 1, 2, \cdots, deg(u) - 1$, in a predetermined manner, where $deg(u)$ is the degree of u. In general, the numbering is not assumed to be consistent, i.e., an edge $(u, v) \in E$ may be the i-th edge of u but the j-th edge of v, where $i \neq j$.

When a robot is in a vertex $u \in V$ it is told the degree $deg(u)$ of u. However, all vertices of the same degree are not distinguishable. The robots are not allowed to put any information such as tokens or markers at the vertices that they visit. At any time step a robot is only allowed to either traverse an edge, or stay in place. When the robot is at a vertex u it may ask to traverse the i-th edge $(u, v) \in E$ of u, where $0 \leq i \leq deg(u)-1$. The robot observes itself at vertex v, the another endpoint of this edge. As described before, the i-th edge of u is the j-th edge of v, for some $0 \leq j \leq deg(v)-1$. In general $j \neq i$.

There are two different variants of the model used in the field. In the first one, the robot is told the index j of the edge it used to enter v. This allows the robot to return to u

* Supported by the Research Council of Norway through the SPECTRUM project.

T. Tokuyama (Ed.): ISAAC 2007, LNCS 4835, pp. 549–560, 2007.

at the next step, if it wants to do so. This variant of the problem is called the rendezvous problem with backtracking. In the second variant of the model, the robot observes itself at vertex v without knowing which edge it used to get there. We call this variant of the problem the general rendezvous problem.

Same as most of work [3,6,9] described, the main strategy is to give the two robots deterministic sequences of instructions which will gurrantee that two robots would eventually meet each other, no matter in which graph they are located, and no matter when they are activated. It is, however, expected that such a meeting would happen as soon as possible. A robot is unaware of the whereabouts of another robot, even it is very close to another one in the graph. The two robots meet only when they are both active and are at same vertex at same time. In particular, the two robots may traverse the same edge but in different directions, and still miss each other.

For the deterministic solutions, it has to be assumed that two robots have different labels, e.g. $L_1 \neq L_2$. Without such an assumption there is no deterministic way of breaking symmetry and no deterministic strategy is possible. An example has been shown in [9] if the two robots are completely identical. Assume G is a ring on n vertices and that the edges are labeled so that out of every vertex, edge 0 goes clockwise, while edge 1 goes anti-clockwise. If the two robots start the same time at different vertices and follow the same instructions, they would never meet! Same as the previous work [9], We also assume that the moves of the two robots are synchronous after both of them are activated. The crutial feature of this problem is that the two robots may be activated at different times which decides arbitrarily by the adversary. A meeting can happen only when both robots are active. The time complexity of any solution is bounded by the number of steps used to complete such a task, which counts from the activation of the second robot.

The *treasure hunt problem* is the variant of the rendezvous problem in which the robots are assigned the labels 0 and 1 and robot 0, the treasure, cannot move, which firstly introduced in [9]. As in the rendezvous problem, the treasure and the seeking robot are not necessarily activated at the same time.

1.1 Previous Work

Dessmark *et al.* [3] presented a deterministic solution of the rendezvous problem which guarantees a meeting of the two robots after a number of steps which is polynomial in n, the size of the graph, l, the length of the shorter of the two labels, and τ, the difference between their activation times. More specifically, the bound on the number of steps that they obtain is $\tilde{O}(n^5\sqrt{\tau l} + n^{10}l)$. In the same paper, Dessmark *et al.* [3] also ask whether it is possible to obtain a polynomial bound that is independent of τ. Kowalski and Malinowski [6] have recently presented a deterministic solution to the rendezvous problem that guarantees a meeting after at most $\tilde{O}(n^{15} + l^3)$ steps, which is independent of τ, and also firstly answer the open problem of [3] when backtracking is allowed. Very recently in [9], Ta-Shma, and Zwick propose a deterministic solution that guarantees a rendezvous within $\tilde{O}(n^5 l)$ time units after the activation of the second robot, and also does not use backtracking. This is the currently best known solution. All the solutions mentioned above rely on the existence of a universal traversal sequences, introduced by Aleliunas *et al.* [1], and are therefore non-explicit. The first explicit

solution for both rendezvous problem and treasure hunt problem can be found in [9]. This work allows backtracking, by using the explicit construction of a *strongly universal exploration sequence SUES*. The time complexity of the solutions for both problems is $O(n^{2c})$, where c is a huge constant. Other variants of the rendezvous problem could be found in [5].

Note if randomization is allowed, then both the rendezvous problem and the treasure hunt problem have the trivial solutions using a polynomial number of steps in term of the size of the graph with high probability, e.g. a random walk by Coppersmith *et al.* [2].

1.2 Our Results

We mainly study here the explicit deterministic treasure hunt problem with backtracking in a n-vertex network. It is the variant of the well known rendezvous problem in which one of the robot (the treasure) is always stationary. We obtain an $O(n^{c(1+\frac{1}{\lambda})})$-time solution for such a problem, which significantly improves currently best known result with running time $O(n^{2c})$ in [9], where c is a fixed constant from the construction of an universal exploration sequence in [8,9], λ is a constant integer and $\lambda \gg 1$. The treasure hunt problem motivates the study of strongly universal exploration sequences. We give a better explicit construction of strongly universal exploration sequences (SUESs) than the one in [9]. The improved explicit *SUESs* could be also used to improve the explicit solution of the rendezvous problem in [9].

2 The Treasure Hunt Problem

The treasure hunt problem is the variant of the well known rendezvous problem in which one of the robots, the treasure, is always stationary. A seeking robot and the treasure are placed in an unknown location in an unknown environment, modeled again by a finite, connected, undirected graph. Same as in the rendezvous problem, the treasure and the seeking robot searching for it are not necessarily activated at the same time. The most difficult case of the problem is when the seeking robot is activated before the treasure.

To clarify our presentation, we give a formal definition for the treasure hunt problem, which is a modified version from the rendezvous problem by Ta-Shma and Zwick [9].

Formally, a deterministic solution for the general treasure hunt problem (without backtracking) is a deterministic algorithm that computes a function $f : Z^+ \times Z^+ \rightarrow Z^+$, where for $d \geq 1$ and $t \geq 0$ we have $0 \leq f(d,t) \leq d-1$. This function defines the walk carried out by the seeking robot as follows: at the t-th time unit since activation, when at a vertex of degree d, use edge number $f(d,t)$ to walk in the next step.

A deterministic solution for the treasure hunt problem with backtracking is a deterministic algorithm that computes a function $f : Z^+ \times Z^+ \times (Z^+)^* \rightarrow Z^+$, where for every $d \geq 1$, $t \geq 0$, and $T \in (Z^+)^*$ we have $0 \leq f(d,t,T) \leq d-1$. This function defines the walk carried out by the seeking robot as follows: at the t-th time unit since activation, when at a vertex of degree d, if the sequence of edge numbers assigned to the edges that were used to enter the vertices at the previous time units is $T \in (Z^+)^*$ the robot will exit the current node using the edge number $f(d,t,T)$, in the next step. In our solutions that use this model, the function f depends on T only through its last element, which is the same as [9].

Throughout most of this paper we shall assume that the graph G in which the robots (seeking robot and the treasure robot) are placed is a d-regular graph, for some $d \geq 3$. Note it is easy to extend the solutions given for the d-regular graphs to general graphs using the ideas from [3].

3 Universal and Strongly Universal Exploration Sequences

For clarity of presentation, we use the same definitions as in [9].

Let $G = (V, E)$ be a d-regular graph. A sequence $\tau_1\tau_2 \cdots \tau_k \in \{0, 1, 2, \cdots, d-1\}^k$ and a starting edge $e_0 = (v_{-1}, v_0) \in E$ define a walk $v_{-1}, v_0, \cdots, v_k$ as follows: For $1 \leq i \leq k$, if (v_{i-1}, v_i) is the s-th edge of v_i, let $e_i = (v_i, v_{i+1})$ be the $(s + \tau_i)$-th edge of v_i, where we assume here that the edges of v_i are numbered $0, 1, \cdots, d-1$, and that $s + \tau_i$ is computed modulo d.

Definition 1. *(Universal Exploration Sequences (UESs) [9]) A sequence $\tau_1\tau_2 \cdots \tau_l \in \{0, 1, \cdots, d-1\}^l$ is a universal exploration sequence for d-regular graphs of size at most n if for every connected d-regular graph $G = (V, E)$ on at most n vertices, any numbering of its edges, and any starting edge $(v_{-1}, v_0) \in E$, the walk obtained visits all the vertices of the graph.*

Reingold [8] obtains an explicit construction of polynomial-size *UES:*

Theorem 1. *([8]) There exists a constant $c \geq 1$ such that for every $d \geq 3$ and $n \geq 1$, a UES of length $O(n^c)$ for d-regular graphs of size at most n can be constructed, deterministically, in polynomial time.*

Definition 2. *(Strongly Universal Exploration Sequences (SUESs) [9]) A possibly infinite sequence $\tau = \tau_1\tau_2 \cdots$, where $\tau_i \in \{0, 1, \cdots, d-1\}$, is a strongly universal exploration sequence (SUES) for d-regular graphs with cover time $p(\cdot)$, if for any $n \geq 1$, any contiguous subsequence of τ of length $p(n)$ is a UES for d-regular graphs of size n.*

Let $O(n^c)$ be the length of a UES (see [8,9]), the main Theorem of this section shows that strongly universal exploration sequences (SUESs) do exist and they can be constructed deterministically in polynomial time with cover time $p(n) = O(n^{c(1+\frac{1}{\lambda})})$, which significantly improves the currently best known result in [9] with $p(n) = O(n^{2c})$, where c is a fixed constant from the construction of an universal exploration sequence in [8,9], λ is a constant integer and $\lambda \gg 1$.

In this section, we firstly give a weak solution with $p(n) = O(n^{\frac{3}{2}c})$, then we show our main result described above.

3.1 Explicit SUESs with $p(n) = O(n^{\frac{3}{2}c})$

In this section, we propose a new explicit strongly universal exploration sequence with cover time $p(n) = O(n^{\frac{3}{2}c})$ for the d-regular graphs of size at most n, where c is the same constant as was used in [8,9]. It is a weak version (a special case) of our main result in Section 3.2, but which gives more intuition on our new approaches in Section 3.2.

Properties of Exploration Sequences. A very useful property of exploration sequences [9] is that walks defined by an exploration sequence can be reversed. For $\tau = \tau_1\tau_2\cdots\tau_k \in \{0,1,\cdots,d-1\}^k$, we let $\tau^{-1} = \tau_k^{-1}\tau_{k-1}^{-1}\cdots\tau_1^{-1}$, where $\tau_i^{-1} = d - \tau_i$. It is not difficult to see that a walk defined by an exploration sequence τ can be backtracked by executing the sequence $0\tau^{-1}0$. Note that if $e_0, e_1, \cdots, e_k$ is the sequence of edges defined by τ, starting with e_0, then executing $0\tau^{-1}0$, starting with e_k defines the sequence $e_k, \tilde{e}_k, \tilde{e}_{k-1}, \cdots, \tilde{e}_0, e_0$, where $\tilde{e}$ is the reverse of edge e. Also, if τ is a universal exploration sequence for graphs with size at most n, then so is $0\tau^{-1}$ starting with the last edge defined by τ.

Construction of SUESs. Let U_n be a sequence of length n which is a universal exploration sequence for d-regular graphs of size at least $bn^{\frac{1}{c}}$, for some constants $b > 1$, and $c > 1$, which can be constructed, deterministically, in polynomial time of n due to Theorem 1. We are interested in sequences U_n only if n is a power of 2. We assume from now that for every $k = 2^i$ and $n = 2^j$, where $i < j$, U_k is a prefix of U_n, e.g., $U_n = U_1U_1U_2U_4\cdots U_{\frac{n}{2}}$.

A strongly universal exploration sequence S_n is a sequence defined in a recursive manner. Our approach is based on the similar idea in [9], but different interleaving components between the symbols which originate from U_n. We begin with $S_1 = U_1$. Assume that $U_n = u_1u_2\cdots u_n$ and that $n \geq 2$.

Define,

$$S_n = u_1S_{r_1}0S_{r_1}^{-1}0u_2S_{r_2}0S_{r_2}^{-1}0u_3\cdots u_iS_{r_i}0S_{r_i}^{-1}0u_{i+1}\cdots u_{n-1}S_{r_{n-1}}0S_{r_{n-1}}^{-1}0u_n,$$

for every $1 \leq i \leq \frac{n}{2}$, we set $r_i = \langle i\rangle$, where $\langle i\rangle = \max\{2^j | 2^{\frac{3}{2}j} \leq i, j \in Z^+\}$. For every $\frac{n}{2} < i < n$, we also assign $r_i = r_{n-i}$. Note that as $n = 2^j$, for some $j \geq 1$, for every $k = 2^p$, where $p < j$, the sequence S_k is a prefix of S_n.

Furthermore, the sequence S_n^{-1} differs with S_n only on the symbols that originate from U_n and in the alignment of the $0'$s:

$$S_n^{-1} = u_n^{-1}0S_{r_1}0S_{r_1}^{-1}u_{n-1}^{-1}\cdots u_{i+1}^{-1}0S_{r_{n-i}}0S_{r_{n-i}}^{-1}u_i^{-1}\cdots u_2^{-1}0S_{r_{n-1}}0S_{r_{n-1}}^{-1}u_1^{-1}.$$

Note that $r_i \leq \sqrt[3]{i^2}$. Thus, if $r_{\frac{n}{2}} = \sqrt[3]{(\frac{n}{2})^2}$, the first half of S_n is equal to $S_{\frac{n}{2}}S_{\sqrt[3]{(\frac{n}{2})^2}}0$, and ends with a full copy of $S_{\sqrt[3]{(\frac{n}{2})^2}}$, followed by a 0. Similarly, the second half of S_n starts with a full copy of $S^{-1}_{\sqrt[3]{(\frac{n}{2})^2}}$. In the following, we bound the length of S_n.

Lemma 1. *For every $n = 2^j$, where $j \geq 1$, $|S_n| < 258n$.*

Proof. Let $s_n = |S_n|$. It is not difficult to see that $s_1 = 1$, $s_2 = 6$, $s_4 = 16$, $s_8 = 46$, $s_{16} = 126$, $s_{32} = 286, \cdots, s_{256} = 3426$. The claim that $s_n \leq 258n$ for every $n \geq 512$ then follows by using simple induction. It is not difficult to see that

$$|S_{2^i}| = |U_{2^i}| + (|U_{2^i}| - 1)\cdot 2(|S_1| + 1) + \sum_{j=1}^{\lfloor\frac{2i}{3}\rfloor - 1} \lfloor\frac{|U_{2^i}| - 1}{2^{\frac{3j}{2}}}\rfloor \cdot 2(|S_{2^j}| - |S_{2^{j-1}}|)$$

$$
\begin{aligned}
s_{2^i} &\le 2^i + 4\cdot 2^i + 2^i \cdot \sum_{j=1}^{\lfloor \frac{2i}{3}\rfloor -1} \frac{2(s_{2^j} - s_{2^{j-1}})}{2^{\frac{3j}{2}}} \\
&= 2^i + 4\cdot 2^i + 2^i \cdot (\sum_{j=1}^{4} \frac{2(s_{2^j} - s_{2^{j-1}})}{2^{\frac{3j}{2}}} + \sum_{j=5}^{\lfloor \frac{2i}{3}\rfloor -1} \frac{2(s_{2^j} - s_{2^{j-1}})}{2^{\frac{3j}{2}}}) \\
&\le 5\cdot 2^i + 2^i \cdot \sum_{j=1}^{4} \frac{2(s_{2^j} - s_{2^{j-1}})}{2^{\frac{3j}{2}}} + 2^i \cdot \sum_{j=5}^{\lfloor \frac{2i}{3}\rfloor -1} \frac{2(s_{2^{j-1}} + 2s_{2^{\frac{2(j-1)}{3}}} + 2)}{2^{\frac{3j}{2}}} \\
&\le 5\cdot 2^i + 2^i \cdot \sum_{j=1}^{4} \frac{2(s_{2^j} - s_{2^{j-1}})}{2^{\frac{3j}{2}}} + 2^i \cdot \sum_{j=5}^{+\infty} \frac{2(s_{2^{j-1}} + 2s_{2^{\frac{2(j-1)}{3}}} + 2)}{2^{\frac{3j}{2}}} \\
&\le 5\cdot 2^i + 2^i \cdot \sum_{j=1}^{4} \frac{2(s_{2^j} - s_{2^{j-1}})}{2^{\frac{3j}{2}}} + 2^i \cdot \sum_{j=5}^{+\infty} \frac{2(s_{2^{j-1}} + 2s_{2^{\frac{2(j-1)}{3}}})}{2^{\frac{3j}{2}}} + 2^i \cdot \sum_{j=5}^{+\infty} \frac{4}{2^{\frac{3j}{2}}} \\
&< 6\cdot 2^i + 2^i \cdot \sum_{j=1}^{4} \frac{2(s_{2^j} - s_{2^{j-1}})}{2^{\frac{3j}{2}}} + 2^i \cdot \sum_{j=5}^{+\infty} \frac{2(s_{2^{j-1}} + 2s_{2^{\frac{2(j-1)}{3}}})}{2^{\frac{3j}{2}}} \\
&\le 6\cdot 2^i + 2^i \cdot (5 + \frac{35\sqrt{2}}{8}) + 2^i \cdot \sum_{j=5}^{+\infty} \frac{2(s_{2^{j-1}} + 2s_{2^{\frac{2(j-1)}{3}}})}{2^{\frac{3j}{2}}} \\
&\le 6\cdot 2^i + 12\cdot 2^i + 2^i \cdot \sum_{j=5}^{+\infty} \frac{2(s_{2^{j-1}} + 2s_{2^{\frac{2(j-1)}{3}}})}{2^{\frac{3j}{2}}} \\
&\le 18\cdot 2^i + 2^i \cdot \sum_{j=5}^{+\infty} \frac{2(258\cdot 2^{j-1} + 2\cdot 258\cdot 2^{\frac{2(j-1)}{3}})}{2^{\frac{3j}{2}}} \\
&\le 18\cdot 2^i + 2^i \cdot 258 \cdot (\frac{2^{-\frac{5}{2}}}{1-\frac{1}{\sqrt{2}}} + \frac{2^{-\frac{17}{6}}}{1-\frac{1}{2^{\frac{5}{6}}}}) \\
&< 18\cdot 2^i + 2^i \cdot 258 \cdot (0.93) \\
&= (257.94)\cdot 2^i \\
&< 258\cdot 2^i.
\end{aligned}
$$

The sequence S_n possesses the following interesting combinatorial property:

Lemma 2. *Let k and $n \ge 2k^{\frac{3}{2}}$ be powers of 2. Then, every subsequence T of S_n or S_n^{-1} of length $s_{2k^{\frac{3}{2}}} + 1 = |S_{2k^{\frac{3}{2}}}| + 1 \le 516k^{\frac{3}{2}}$ contains, as a contiguous subsequence, S_k or $0S_k^{-1}$.*

Proof. We prove the claim by induction on n. If $n = 2k^{\frac{3}{2}}$ then the claim is vacuously satisfied as S_n contains a full S_k.

Assume, therefore, that the claim holds for every $m = 2^{j'}$ that satisfies $2k^{\frac{3}{2}} \le m < n = 2^j$. We show that it also holds for n. Let T be a subsequence of S_n of length $s_{2k^{\frac{3}{2}}} + 1$. Essentially the same argument works if T is a subsequence of such length

of S_n^{-1}. We use the **exactly same** arguments as in Lemma 6.2 [9]. For completeness of our presentaion, we reproduce the analysis from [9].

We consider the following cases:

Case 1: T is completely contained in a subsequence S_m or S_m^{-1} of S_n, for some $m < n$.

The claim then follows immediately from the induction hypothesis.

Case 2: T is completely contained in a subsequence $S_m 0 S_m^{-1}$ of S_n, for some $m < n$.

In this case, $T = T'0T''$, where T' is a suffix of S_m and T'' is a prefix of S_m^{-1}. Either $|T'| \geq \frac{1}{2}s_{2k^{\frac{3}{2}}}$ or $|T''| \geq \frac{1}{2}s_{2k^{\frac{3}{2}}}$. Assume that $|T''| \geq \frac{1}{2}s_{2k^{\frac{3}{2}}}$. Another case is analogous. As T'' is a prefix of S_m^{-1}, and $|T''| \geq \frac{1}{2}s_{2k^{\frac{3}{2}}}$, it follows that $m \geq 2k^{\frac{3}{2}}$. Now, S_k^{-1} is almost a prefix of S_m^{-1}, in the sense that they differ only in symbols that originate directly from S_m. In particular, a prefix of S_m^{-1} of length $\frac{1}{2}s_{2k^{\frac{3}{2}}}$, half the length of $S_{2k^{\frac{3}{2}}}$, ends with a full copy of S_k, followed by 0.

Case 3: T contains a symbol u_l of S_n that originates from U_n.

In this case, $T = T'u_lT''$. Again, we have either $|T'| \geq \frac{1}{2}s_{2k^{\frac{3}{2}}}$ or $|T''| \geq \frac{1}{2}s_{2k^{\frac{3}{2}}}$. Assume again that $|T''| \geq \frac{1}{2}s_{2k^{\frac{3}{2}}}$. Another case is analogous. Let

$$S_{n,l} = u_l S_{r_l} 0 S_{r_l}^{-1} 0 u_{l+1} \cdots u_{n-1} S_{r_{n-1}} 0 S_{r_{n-1}}^{-1} 0 u_n$$

be the suffix of S_n that starts with the symbol u_l that originates from the l-th symbol of U_n. We claim that the prefix of $S_{n,l}$ of length $\frac{1}{2}s_{2k^{\frac{3}{2}}}$ contains a copy of S_k. Let $l' = \lceil \frac{l}{k^{\frac{3}{2}}} \rceil k^{\frac{3}{2}}$ be the first index after l which is divisible by $k^{\frac{3}{2}}$. Clearly $r_{l'} \geq k$ and hence S_k is a prefix of $S_{r_{l'}}$. Thus, $S' = u_l S_{r_l} 0 S_{r_l}^{-1} 0 \cdots u_{l'} S_k$ is a prefix of $S_{m,l}$ which ends with a complete S_k. As for every $l \leq i < l'$ we have $r_i = r_{i \bmod k^{\frac{3}{2}}}$, we have that S' is contained in the first half of $S_{2k^{\frac{3}{2}}}$, and hence $|S'| \leq \frac{1}{2}s_{2k^{\frac{3}{2}}}$ as expected.

We are now ready to prove the following Theorem.

Theorem 2. *If for any $n \geq 1$ of the power of 2 there exists an UES of length $O(n^c)$ for a d-regular graph of size at most n, then there is an infinite SUES for this d-regular graph with cover time $p(n) = O(n^{\frac{3}{2}c})$, where c is the fixed constant from the construction of an universal exploration sequence in [8,9]. Furthermore, the SUESs can be constructed deterministically in polynomial time.*

Proof. Let us look at the recursive definition of S_n and ignore all the recursive components of S_j such as $j < n$, and their inverses, which because that $0S_j^{-1}0$ reverses the actions of S_j. The left parts are $U_n = u_1, u_2, u_3, \cdots, u_n$. However, note that U_n is a UES for for d-regular graphs of size at least $bn^{\frac{1}{c}}$, for some constants $b \geq 1$ and $c > 1$ due to Theorem 1. Theorem 1 also show that such a UES can be constructed, deterministically, in polynomial time. According to Lemma 2, we know that every subsequence T of the SUES we constructed of length $s_{2n^{\frac{3}{2}}} + 1 = O(n^{\frac{3}{2}})$ contains, as a contiguous subsequence, a full copy of S_n. Consequently, there is an infinite SUES for d-regular graphs with cover time $p(n) = O(n^{\frac{3}{2}c})$, where c is the fixed constant from the construction of an universal exploration sequence in [8,9]. Furthermore, the SUESs can be constructed deterministically in polynomial time.

This thus gives us an explicit solution to the treasure hunt problem. In fact, the seeking robot just need run the SUES. The adversary will decide when the treasure is put into the graph. But note that the subsequence of a SUES with length $p(n)$ starting at the activation point forms a UES, and then the seeking robot finds the treasure by following the instruction of the sequence.

3.2 Explicit SUESs with $p(n) = O(n^{c(1+\frac{1}{\lambda})})$

In this section, we propose our main result, a new explicit strongly universal exploration sequence with cover time $p(n) = O(n^{c(1+\frac{1}{\lambda})})$, which significantly improves the currently best known result in [9] with $p(n) = O(n^{2c})$, where c is the fixed constant from the construction of an universal exploration sequence in [8,9], λ is a constant integer and $\lambda \gg 1$.

Treasure Sequences. A treasure sequence is an infinite sequence $Q(\lambda) = q_1, q_2, q_3, q_4, \cdots$, based on a constant integer $\lambda \gg 1$ such as

$$q_i = \sum_{j=i}^{+\infty} (2^{\frac{-j}{\lambda}} + 2^{-(\frac{2\lambda+1}{\lambda^2+\lambda}\cdot j + \frac{\lambda}{\lambda+1} - 2)} + 2^{-(\frac{\lambda+1}{\lambda}\cdot j - 2)}),$$

where $i \geq 1$. It is easy to see that the treasure sequences $Q(\lambda)$ are monotonely decreasing, $\lim_{i\to+\infty} q_i = 0$. We call the first element or term $q_i < 1$ in $Q(\lambda)$ a *golden ball*, where $i \geq 1$. Similarly, we call the fixed index i of the *golden ball* of $Q(\lambda)$ as the *golden point*, where λ is a fixed integer constant and $\lambda \gg 1$.

The following Lemma follows directly.

Lemma 3. *There exists a golden ball in the treasure sequence.*

The property of the treasure sequence will be used to further reduce the length of the cover time $p(n)$ of the SUESs later.

Construction of SUESs. Same as in Section 3.1, let U_n be a sequence of length n which is a UES for d-regular graphs of size at least $bn^{\frac{1}{c}}$, for some constants $b \geq 1$, and $c > 1$. And for every $k = 2^i$ and $n = 2^j$, where $i < j$, U_k is a prefix of U_n.

We now define recursively a sequence S_n of strongly universal exploration sequences. We start with $S_1 = U_1$. Assume that $U_n = u_1 u_2 \cdots u_n$ and that $n \geq 2$. Define,

$$S_n = u_1 S_{r_1} 0 S_{r_1}^{-1} 0 u_2 S_{r_2} 0 S_{r_2}^{-1} 0 u_3 \cdots u_i S_{r_i} 0 S_{r_i}^{-1} 0 u_{i+1} \cdots u_{n-1} S_{r_{n-1}} 0 S_{r_{n-1}}^{-1} 0 u_n,$$

where for every $1 \leq i \leq \frac{n}{2}$, we set $r_i = \langle\langle i \rangle\rangle$, where $\langle\langle i \rangle\rangle = max\{2^j | 2^{\frac{\lambda+1}{\lambda}j} \leq i, j \in Z^+\}$. Also for every $\frac{n}{2} < i < n$, we assign $r_i = r_{n-i}$. Note that as $n = 2^j$, for some $j \geq 1$, for every $k = 2^p$, where $p < j$, the sequence S_k is a prefix of S_n.

Note that $r_i \leq \sqrt[\lambda+1]{i^\lambda}$. Thus, if $r_{\frac{n}{2}} = \sqrt[\lambda+1]{(\frac{n}{2})^\lambda}$, the first half of S_n is equal to $S_{\frac{n}{2}} S_{\sqrt[\lambda+1]{(\frac{n}{2})^\lambda}} 0$, and ends with a full copy of $S_{\sqrt[\lambda+1]{(\frac{n}{2})^\lambda}}$, followed by a 0. Similarly, the second half of S_n starts with a full copy of $S^{-1}_{\sqrt[\lambda+1]{(\frac{n}{2})^\lambda}}$.

We next bound the length of S_n.

Let $q_g^{[\lambda]}, g$ denote the *golden ball* and *golden point* of the treasure sequence $Q(\lambda)$, respectively. Further more, if $g \geq 2$, we set $C_{finite} = \sum_{j=1}^{g-1} \frac{2(|S_{2^j}|-|S_{2^{j-1}}|)}{2^{\frac{(\lambda+1)j}{\lambda}}}$, and $C_{max} = max\{y | y = \frac{S_{2^j}}{2^j},\ for\ 1 \leq j \leq g-1\}$, otherwise $C_{finite} = C_{max} = 0$ (e.g. $g = 1$). Consequently, $q_g^{[\lambda]}, g, C_{finite}$, and C_{max} are constants due to the definition of the treasure sequence and the fact that only a finite number of terms are involved in the calculations, where $0 < q_g^{[\lambda]} < 1$, and $g \geq 1$.

We are now ready to prove the following Lemma.

Lemma 4. *For every $n = 2^j$, where $j \geq 1$, $|S_n| < (\frac{5+C_{finite}}{1-q_g^{[\lambda]}} + C_{max})n$.*

Proof. Let $s_n = |S_n|$. For every $n \leq 2^{g-1}$, we know it is true due to the definition of the constant C_{max}. The claim that $s_n < (\frac{5+C_{finite}}{1-q_g^{[\lambda]}} + C_{max})n$ for every $n \geq 2^g$ then follows by using the induction. It is not difficult to see that

$$
\begin{aligned}
|\ _{2^i}| &= |\ _{2^i}| + (|\ _{2^i}| - 1) \cdot 2(|\ _1| + 1) + \sum_{j=1}^{\lfloor \frac{\lambda}{\lambda+1} \rfloor \cdot i - 1} \lfloor \frac{|\ _{2^i}| - 1}{2^{\frac{(\lambda+1)j}{\lambda}}} \rfloor \cdot 2(|\ _{2^j}| - |\ _{2^{j-1}}|) \\
\ _{2^i} &\leq 2^i + 4 \cdot 2^i + 2^i \cdot \sum_{j=1}^{\lfloor \frac{\lambda}{\lambda+1} \rfloor \cdot i - 1} \frac{2(\ _{2^j} - \ _{2^{j-1}})}{2^{\frac{(\lambda+1)j}{\lambda}}} \\
&= 2^i + 4 \cdot 2^i + 2^i \cdot \sum_{j=1}^{g-1} \frac{2(\ _{2^j} - \ _{2^{j-1}})}{2^{\frac{(\lambda+1)j}{\lambda}}} + 2^i \cdot \sum_{j=g}^{\lfloor \frac{\lambda}{\lambda+1} \rfloor \cdot i - 1} \frac{2(\ _{2^j} - \ _{2^{j-1}})}{2^{\frac{(\lambda+1)j}{\lambda}}} \\
&\quad 5 \cdot 2^i + 2^i \cdot \sum_{j=1}^{g-1} \frac{2(\ _{2^j} - \ _{2^{j-1}})}{2^{\frac{(\lambda+1)j}{\lambda}}} + 2^i \cdot \sum_{j=g}^{+\infty} \frac{2(\ _{2^j} - \ _{2^{j-1}})}{2^{\frac{(\lambda+1)j}{\lambda}}} \\
&\leq 5 \cdot 2^i + \ _{finite} \cdot 2^i + 2^i \cdot \sum_{j=g}^{+\infty} \frac{2(\ _{2^{j-1}} + 2\ _{2^{\frac{(j-1)\lambda}{\lambda+1}}} + 2)}{2^{\frac{(\lambda+1)j}{\lambda}}} \\
&\quad 5 \cdot 2^i + \ _{finite} \cdot 2^i + 2^i \cdot (\frac{5 + \ _{finite}}{1 - \ _g^{[\lambda]}} + \ _{max}) \cdot \sum_{j=g}^{+\infty} \frac{2(2^{j-1} + 2 \cdot 2^{\frac{(j-1)\lambda}{\lambda+1}} + 2)}{2^{\frac{(\lambda+1)j}{\lambda}}} \\
&= 5 \cdot 2^i + \ _{finite} \cdot 2^i + 2^i \cdot (\frac{5 + \ _{finite}}{1 - \ _g^{[\lambda]}} + \ _{max}) \cdot \ _g^{[\lambda]} \\
&\quad 5 \cdot 2^i + \ _{finite} \cdot 2^i + 2^i \cdot \ _{max} \cdot (1 - \ _g^{[\lambda]}) + 2^i \cdot (\frac{5 + \ _{finite}}{1 - \ _g^{[\lambda]}} + \ _{max}) \cdot \ _g^{[\lambda]} \\
&= 2^i \cdot (\frac{5 + \ _{finite}}{1 - \ _g^{[\lambda]}} + \ _{max})
\end{aligned}
$$

Using the same arguments as in Lemma 2, we can prove the following Lemma.

Lemma 5. *Let k and $n \geq 2k^{\frac{\lambda+1}{\lambda}}$ be powers of 2. Then, every subsequence T of S_n or S_n^{-1} of length $s_{2k^{\frac{\lambda+1}{\lambda}}} + 1 = O(k^{\frac{\lambda+1}{\lambda}})$ contains, as a contiguous subsequence, a full of S_k or $0S_k^{-1}$.*

Proof. We prove the claim by induction on n. If $n = 2k^{\frac{\lambda+1}{\lambda}}$ then the claim is vacuously satisfied as S_n contains a full S_k.

Assume, therefore, that the claim holds for every $m = 2^{j'}$ that satisfies $2k^{\frac{\lambda+1}{\lambda}} \leq m < n = 2^j$. We show that it also holds for n. Let T be a subsequence of S_n of length $s_{2k^{\frac{\lambda+1}{\lambda}}} + 1$. Essentially the same argument works if T is a subsequence of such length of S_n^{-1}.

Same as in Lemma 2, we study the following cases:

Case 1: T is completely contained in a subsequence S_m or S_m^{-1} of S_n, for some $m < n$.

The claim then follows immediately from the induction hypothesis.

Case 2: T is completely contained in a subsequence $S_m 0 S_m^{-1}$ of S_n, for some $m < n$.

In this case, $T = T'0T''$, where T' is a suffix of S_m and T'' is a prefix of S_m^{-1}. Either $|T'| \geq \frac{1}{2}s_{2k^{\frac{\lambda+1}{\lambda}}}$ or $|T''| \geq \frac{1}{2}s_{2k^{\frac{\lambda+1}{\lambda}}}$. Assume that $|T''| \geq \frac{1}{2}s_{2k^{\frac{\lambda+1}{\lambda}}}$. Another case is analogous. As T'' is a prefix of S_m^{-1}, and $|T''| \geq \frac{1}{2}s_{2k^{\frac{\lambda+1}{\lambda}}}$, it follows that $m \geq 2k^{\frac{\lambda+1}{\lambda}}$. Now, S_k^{-1} is almost a prefix of S_m^{-1}, in the sense that they differ only in symbols that originate directly from S_m. In particular, a prefix of S_m^{-1} of length $\frac{1}{2}s_{2k^{\frac{\lambda+1}{\lambda}}}$, half the length of $S_{2k^{\frac{\lambda+1}{\lambda}}}$, ends with a full copy of S_k, followed by 0.

Case 3: T contains a symbol u_l of S_n that originates from U_n.

In this case, $T = T'u_lT''$. Again, we have either $|T'| \geq \frac{1}{2}s_{2k^{\frac{\lambda+1}{\lambda}}}$ or $|T''| \geq \frac{1}{2}s_{2k^{\frac{\lambda+1}{\lambda}}}$. Assume again that $|T''| \geq \frac{1}{2}s_{2k^{\frac{\lambda+1}{\lambda}}}$. Another case is analogous. Let

$$S_{n,l} = u_l S_{r_l} 0 S_{r_l}^{-1} 0 u_{l+1} \cdots u_{n-1} S_{r_{n-1}} 0 S_{r_{n-1}}^{-1} 0 u_n$$

be the suffix of S_n that starts with the symbol u_l that originates from the l-th symbol of U_n. We claim that the prefix of $S_{n,l}$ of length $\frac{1}{2}s_{2k^{\frac{\lambda+1}{\lambda}}}$ contains a copy of S_k. Let $l' = \lceil \frac{l}{k^{\frac{\lambda+1}{\lambda}}} \rceil k^{\frac{\lambda+1}{\lambda}}$ be the first index after l which is divisible by $k^{\frac{\lambda+1}{\lambda}}$. Clearly $r_{l'} \geq k$ and hence S_k is a prefix of $S_{r_{l'}}$. Thus, $S' = u_l S_{r_l} 0 S_{r_l}^{-1} 0 \cdots u_{l'} S_k$ is a prefix of $S_{m,l}$ which ends with a complete S_k. As for every $l \leq i < l'$ we have $r_i = r_{i \bmod k^{\frac{\lambda+1}{\lambda}}}$, we have that S' is contained in the first half of $S_{2k^{\frac{\lambda+1}{\lambda}}}$, and hence $|S'| \leq \frac{1}{2}s_{2k^{\frac{\lambda+1}{\lambda}}}$ as expected.

Furthermore, by the same arguments as in Theorem 2, we have:

Theorem 3. *If for every $n \geq 1$ of the power of 2 there is a UES of length $O(n^c)$ for d-regular graphs of size at most n, then there is an infinite SUES for d-regular graphs with cover time $p(n) = O(n^{c(1+\frac{1}{\lambda})})$, where c is the fixed constant from the construction of an universal exploration sequence in [8,9], λ is a constant integer and $\lambda \gg 1$. Furthermore, the SUESs can be constructed deterministically in polynomial time.*

Proof. Let us ignore all the recursive components of S_j from S_n such as $j < n$, and their inverses, which because that $0S_j^{-1}0$ reverses the actions of S_j. The left parts are

$U_n = u_1, u_2, u_3, \cdots, u_n$. Moreover, note that U_n is a UES for for d-regular graphs of size at least $bn^{\frac{1}{c}}$, for some constants $b \geq 1$ and $c > 1$ due to Theorem 1. Theorem 1 also show that such a UES could be constructed, deterministically, in polynomial time. According to Lemma 5, we know that every subsequence T of the SUES we constructed of length $s_{2n^{\frac{\lambda+1}{\lambda}}} + 1 = O(n^{\frac{\lambda+1}{\lambda}})$ contains, as a contiguous subsequence, a full copy of S_n. Consequently, there is an infinite SUES for d-regular graphs with cover time $p(n) = O(n^{\frac{\lambda+1}{\lambda}c})$, where c is the fixed constant from the construction of an universal exploration sequence in [8,9]. Furthermore, the SUESs can be constructed deterministically in polynomial time.

Finally, by employing the standard double techniques in d-regular graphs of size at most n, we get the desired result.

Theorem 4. *If for every $n \geq 1$ there is a UES of length $O(n^c)$ for d-regular graphs of size at most n, then there is an infinite SUES for d-regular graphs with cover time $p(n) = O(n^{c(1+\frac{1}{\lambda})})$, where c is the fixed constant from the construction of an universal exploration sequence in [8,9], λ is a constant integer and $\lambda \gg 1$. Furthermore, the SUESs can be constructed deterministically in polynomial time.*

Remark 1. *It is easy to extend the solutions given for the d-regular graphs to general graphs using the ideas from [3,9]. More details would appear in the full version of the paper.*

4 Conclusion and Open Problems

We obtained improved explicit deterministic solutions for the treasure hunt problem with backtracking. More percisely, we presented an $O(n^{c(1+\frac{1}{\lambda})})$-time solution for the treasure hunt, which significantly improves the currently best known result with running time $O(n^{2c})$ in [9], where c is the fixed constant from the construction of an universal exploration sequence in [8,9], λ is a constant integer and $\lambda \gg 1$. We also give a better explicit construction of strongly universal exploration sequences (SUESs) than the one in [9]. The improved explicit *SUESs* could be also used to improve the explicit solution of the rendezvous problem with backtracking in [9].

The existence of strongly universal exploration sequences without backtracking is left as an intriguing open problem.

Acknowledgments

We thank Fredrik Manne for useful suggestions on the presentation. We are also deeply grateful to the anonymous referees for carefully proofreading the preliminary version of this paper and the helpful remarks.

References

1. Aleliunas, R., Karp, R.M., Lipton, R.J., Lovasz, L., Rackoff, C.: Random walks, universal traversal sequences, and the complexity of maze problems. In: Proceedings of the 20th Annual Symposium on Foundations of Computer Science. FOCS, pp. 218–223 (1979)

2. Coppersmith, D., Tetali, P., Winkler, P.: Collisions among random walks on a graph. SIAM Journal on Discrete Mathematics 6(3), 363–374 (1993)
3. Dessmark, A., Fraigniaud, P., Kowalski, D., Pelc, A.: Deterministic rendezvous in graphs. Algorithmica, 46(1), 69–96 (2006)
4. Dessmark, A., Fraigniaud, P., Pelc, A.: Deterministic rendezvous in graphs. In: Proceedings of 11th Annual European Symposium on Algorithms, ESA, pp. 184–195 (2003)
5. De Marco, G., Gargano, L., Kranakis, E., Krizanc, D., Pelc, A., Vaccaro, U.: Asynchronous deterministic rendezvous in graphs. Theoretical Computer Science, 355(3), 315–326 (2006)
6. Kowalski, D.R., Malinowski, A.: How to meet in anonymous network. In: Flocchini, P., Gkasieniec, L. (eds.) SIROCCO 2006. LNCS, vol. 4056, pp. 44–58. Springer, Heidelberg (2006)
7. Kowalski, D.R., Pelc, A.: Polynomial deterministic rendezvous in arbitrary graphs. In: Fleischer, R., Trippen, G. (eds.) ISAAC 2004. LNCS, vol. 3341, pp. 644–656. Springer, Heidelberg (2004)
8. Reingold, O.: Undirected ST-connectivity in logspace. In: Proceedings of the 37th Annual ACM Symposium on Theory of Computing, STOC, pp. 376–385 (2005)
9. Ta-Shma, A., Zwick, U.: Deterministic rendezvous, treasure hunts and strongly universal exploration sequences. In: Proceedings of the 18th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA, pp. 599–608 (2007)

Hardness and Approximation of Traffic Grooming*

Omid Amini[1,2], Stéphane Pérennes[1], and Ignasi Sau[1,3]

[1] Mascotte joint Project I3S (CNRS/UNSA) and INRIA - 2004, route des Lucioles - Sophia-Antipolis, France
[2] École Polytechnique - Palaiseau, France
[3] Graph Theory and Combinatorics group at Applied Mathematics IV Department of UPC - Barcelona, Spain
FirstName.LastName@sophia.inria.fr

Abstract. Traffic grooming is a central problem in optical networks. It refers to pack low rate signals into higher speed streams, in order to improve bandwidth utilization and reduce network cost. In WDM networks, the most accepted criterion is to minimize the number of electronic terminations, namely the number of SONET Add-Drop Multiplexers (ADMs). In this article we focus on ring and path topologies. On the one hand, we provide the first inapproximability result for Traffic Grooming for fixed values of the grooming factor g, answering affirmatively the conjecture of Chow and Lin (*Networks, 44 :194-202, 2004*). More precisely, we prove that Ring Traffic Grooming for fixed $g \geq 1$ and Path Traffic Grooming for fixed $g \geq 2$ are APX-complete. That is, they do not accept a PTAS unless P = NP. Both results rely on the fact that finding the maximum number of edge-disjoint triangles in a graph (and more generally cycles of length $2g+1$ in a graph of girth $2g+1$) is APX-complete.

On the other hand, we provide a polynomial-time approximation algorithm for Ring and Path Traffic Grooming, based on a greedy cover algorithm, with an approximation ratio independent of g. Namely, the approximation guarantee is $\mathcal{O}(n^{1/3}\log^2 n)$ for any $g \geq 1$, n being the size of the network. This is useful in practical applications, since in backbone networks the grooming factor is usually greater than the network size. As far as we know, this is the first approximation algorithm with this property. Finally, we improve this approximation ratio under some extra assumptions about the request graph.

Keywords : Approximation Algorithms, Traffic Grooming, Optical Networks, SONET ADM, APX-hardness, PTAS, inapproximability.

* This work has been partially supported by European project IST FET AEOLUS, Ministerio de Educación y Ciencia of Spain, European Regional Development Fund under project TEC2005-03575, Catalan Research Council under project 2005SGR00256, and COST action 293 GRAAL, and has been done in the context of the crc Corso with France Telecom.

T. Tokuyama (Ed.): ISAAC 2007, LNCS 4835, pp. 561–573, 2007.

1 Introduction

1.1 Background and Problem Definition

Optical wavelength division multiplexing (WDM) is today the most promising technology to accommodate the explosive growth of Internet and telecommunication traffic in wide-area, metro-area, and local-area networks. Using WDM, the potential bandwidth of 50 THz of a fiber can be divided into multiple non-overlapping wavelength or frequency channels. Since currently the commercially available optical fibers can support over a hundred frequency channels, such a channel has over one gigabit-per-second transmission speed. However, the network is usually required to support traffic connections at rates that are lower than the full wavelength capacity. In order to save equipment cost and improve network performance, it turns out to be very important to aggregate the multiple low-speed traffic connections, namely *requests*, into higher speed streams. Traffic grooming is the term used to carry out this aggregation, while optimizing the equipment cost. In WDM optical networks the most accepted criterion is to minimize the number of electronic terminations, which is unanimously considered as the dominant cost, rather than the number of wavelengths.

SONET ring is the most widely used optical network infrastructure today. In these networks, a communication between a pair of nodes is done via a *lightpath*, and each lightpath uses an Add-Drop Multiplexer (ADM), i.e. an electronic termination, at each of its two endpoints. If each request uses $\frac{1}{g}$ of the capacity of a wavelength, g is said to be the *grooming factor*. The problem is equivalent to assigning a wavelength to each request in such a way that for any wavelength and any link of the network, there can be at most g requests using this link on this wavelength. The aim is to minimize the total number of ADMs. In the graph-theoretical approach that we use, the set of requests is modeled by a graph R, and each vertex in the subgraph of R corresponding to a wavelength represents an ADM. The problem, in the case where the communication network is a ring, can be formally stated as follows :

Ring Traffic Grooming
Input : A cycle C_n on n vertices (network), a graph R (set of requests) on vertices of C_n, and a grooming factor g.
Output : Find for each edge $r = \{x, y\}$ of R, a path $P(r)$ in C_n between x and y, and a partition of the edges of R into subgraphs R_ω, $1 \leq \omega \leq W$, such that for each edge e in $E(C_n)$ and for all ω, the number of paths $P(r)$ using e, r being an edge of R_ω, is at most g.
Objective : Minimize $\sum_{\omega=1}^{W} |V(R_\omega)|$.

The statement of Path Traffic Grooming is analogous, replacing C_n by P_n. To fix ideas, consider a ring on five nodes and the complete graph of Fig. 1 as request graph, and let $g = 2$. We exhibit two valid solutions of the problem, both using two subgraphs (i.e. two wavelengths). The second solution is better because it uses 9 vertices instead of 10.

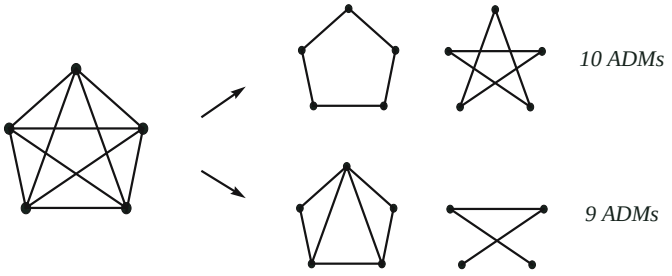


Fig. 1. Two valid partitions of K_5 when $g = 2$, using different number of ADMs

1.2 Previous Work and Our Contribution

The notion of traffic grooming was introduced in [13] for the ring topology. Since then, traffic grooming has been widely studied in the literature (cf. [9], [18], [21] for some surveys). The problem has been proved to be NP-complete for ring networks and general g [5]. Many heuristics have been done [8], but exact solutions have been found only for certain values of g and for the uniform all-to-all traffic case in unidirectional ring [3], bidirectional ring [4], and path topologies [3]. On the other hand, there was no result on the inapproximability of the problem for fixed $g \geq 1$. In [6] the authors conjecture that Traffic Grooming is Max SNP-hard (or equivalently, APX-hard, modulo PTAS-reductions) for any fixed value of the grooming factor. We answer affirmatively to this question in Theorem 2, providing the first hardness result for the Ring Traffic Grooming problem for fixed values of the grooming factor g.

Considering g as part of the input, in [15] it was proved that Path Traffic Grooming does not accept a constant-factor approximation unless P = NP. For fixed values of g, Path Traffic Grooming was proved to be in P for $g = 1$ [3], but the complexity for fixed $g \geq 2$ has been an open question for a while. Recently, it has been proved in [19] that Path Traffic Grooming for fixed $g > 1$ is NP-complete for *bounded number of wavelengths*. Our method permits us to improve this result in Sect. 3, by proving the APX-completeness of Path Traffic Grooming for any fixed $g > 1$ and unbounded number of wavelengths. In particular, this extends the NP-completeness result of [19] to the case where the number of wavelengths is not bounded.

The main ingredient of our approach is the proof of the APX-completeness (given in Sect. 2) of the problem of finding the maximum number of edge-disjoint triangles in a graph with bounded degree B : Maximum Bounded Edge Covering by Triangles (MECT-B for short). The proof is obtained by L-reduction from Maximum Bounded Covering by 3-Sets, which was proved to be MAX SNP-complete in [16]. A simple modification of this technique permits us to prove the APX-completeness of finding the maximum number of edge-disjoint odd cycles of given length in a graph. This later claim is then used to extend our results to arbitrary values of g, see Sections 2, 3 and [1].

The design of approximation algorithms for Traffic Grooming is the topic of the second part of this paper. We present the results for the ring topology, but the same algorithm works also for the path topology. As we show in Sect. 3, it is trivial to obtain a $\mathcal{O}(\sqrt{g})$-approximation with running time polynomial in g and n. For $g = 1$, the best algorithm in rings achieves an approximation ratio of $10/7$ [10]. For general g, the best approximation algorithm [12] achieves an approximation factor of $\mathcal{O}(\log g)$, but the problem is that the running time is exponential in g (that is, n^g). Since in practical applications SONET WDM rings are widely used as backbone optical networks [9], [18], the grooming factor is usually greater than the size of the network, i.e. $g \geq n$. For those networks, the running time of this algorithm becomes exponential in n. Thus, it turns out to be important to find good approximation algorithms with running time polynomial in both n and g. In Sect. 4 we provide such an approximation algorithm, considering g as part of the input. Our algorithm finds a solution of Ring Traffic Grooming that approximates the optimal value within a factor $\mathcal{O}(n^{1/3} \log^2 n)$ for any $g \geq 1$. To the best of our knowledge, this is the first polynomial-time approximation algorithm for the Ring Traffic Grooming problem with an approximation ratio which does not depend on g. Although the performance of this algorithm seems not to be very good at first sight, in fact we conjecture that for the general instance of the problem it is not possible to get rid of a factor n^δ, for some constant $\delta > 0$. Finally, we show that the general scheme of the algorithm yields a $\mathcal{O}(\log^2 n)$-approximation if the request graph excludes a fixed graph as minor, for example if R is planar or of bounded genus. The main theoretical contribution of the second part of this paper is to relate the Traffic Grooming problem to the Dense k-Subgraph problem [11]. We conclude by proposing some further research directions to better understand the complexity of Traffic Grooming.

2 APX-Completeness of MECT-B

In complexity theory, the class APX (Approximable) stands for all NP-hard optimization problems that can be approximated within a constant factor. The subclass PTAS (Polynomial Time Approximation Scheme) contains the problems that can be approximated in polynomial time within a ratio $1 + \varepsilon$ for *all* constants $\varepsilon > 0$. Intuitively, these problems are the easiest ones among all NP-complete problems. Since, assuming P $\neq$ NP, there is a strict inclusion of PTAS into APX (for instance, Vertex Cover $\in$ APX $\setminus$ PTAS), an APX-hardness result for a problem implies the non-existence of a PTAS. MECT-B has been proved to be NP-complete [14], and the APX-hardness when requiring node-disjoint triangles was proved in [16]. The proof of the APX-hardness of MECT-B that we provide can be extended to obtain the APX-completeness of the problem of finding the maximum number of edge-disjoint cycles of length $2g + 1$ for any fixed $g \geq 1$, as it is shown in [1]. For convenience, we prove the Max SNP-hardness of MECT-B, which is known to be the same as the

APX-hardness modulo PTAS-reductions. MECT-B is trivially in APX, since a simple greedy algorithm provides a 3-approximation.

Theorem 1. *(a)* MECT-B, $B \geq 10$ *is* Max SNP*-complete. Furthermore, the problem remains* Max SNP*-complete in tripartite graphs.*

(b) More generally, given a $(2g+1)$*-partite graph* G *of girth* $2g+1$*, consisting of* $(2g+1)$ *parts* $A_0, \ldots, A_{2g}$ *such that the only edges are between* A_i *and* $A_{i+1 \pmod{2g+1}}$*,* $i = 0, \ldots, 2g$*, and such that all the graphs induced by* $V(G) \setminus A_i$ *in* G*, for all* $i = 0, \ldots, 2g$*, form a forest, the problem of finding the maximum number of edge disjoint* C_{2g+1}*'s in* G *is* APX*-complete.*

Proof: We give the proof of part *(a)*, the proof of part *(b)* is given in [1]. L-reduction from Max 3SC-B[1] and L-reduction to Indep. Set-B[2] :

We define h : MECT-B $\rightarrow$ Indep. Set - (3/2(B-2)) as follows : given a graph G as instance I of MECT-B, we define the following instance $h(I)$ of Indep. Set - (3/2(B-2)) : the graph $h(G)$ contains a node v_T for every triangle T in G. There is an edge $\{v_{T_0}, v_{T_1}\}$ in $h(G)$ iff T_0 and T_1 share an edge in G. Given a solution A of $h(I)$, we define a solution $S_h(A)$ of I by taking the triangles corresponding to nodes in A. It is easily verified that (h, S_h) is an L-reduction.

Now, we define f : Max 3SC-B $\rightarrow$ MECT-(3B+1) in the following way : suppose that we are given as instance I, a collection C of 3-element subsets of a set X such that every element of X belongs to at most B members of C. The problem for I consists in finding the maximal number $OPT(I)$ of disjoint subsets in C. We construct an instance $f(I)$ of MECT-(3B+1), i.e. we construct a graph $G = (V, E)$ in which we ask for the maximum number $OPT(f(I))$ of edge-disjoint triangles. Let $C = \{c_1, \ldots, c_r\}$, with $|c_i| = 3$. The local replacement f substitutes for each element $c_i = \{x, y, z\} \in C$, the graph $G_i = (V_i, E_i)$ depicted in Fig. 2.

To avoid confusion, note by t any element in c_i, i.e. $t \in \{x, y, z\}$. Note that, for each element t, the nodes $t[0]$ and $t[1]$, and the edge $t[0]t[1]$ (corresponding

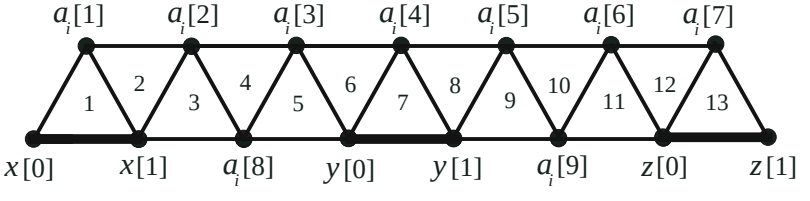


Fig. 2. Gadget used in the reduction of the proof of Theorem 1

[1] Maximum Bounded Covering by 3-Sets : Given a collection of 3-subsets of a given set, each element appearing in at most B subsets, find the maximum number of disjoint subsets.

[2] Maximum Bounded Independent Set : Given a graph of maximum degree $\leq B$, find a maximum independent set.

to the thick edges in Fig. 2) appear only once in G, regardless of the number of occurrences of t. On the other hand, we add 9 new vertices $a_i[j]$, $1 \leq j \leq 9$ for each subset c_i, $1 \leq i \leq |C|$. More precisely, $G = (V, E) = \cup_{i=1}^{|C|} G_i$, where $V = \bigcup_{t \in X} \{t[0], t[1]\} \cup \bigcup_{i=1}^{|C|} \{a_i[j] : 1 \leq j \leq 9\}$ and $E = \bigcup_{i=1}^{|C|} E_i$.

Now, given a solution A of $f(I)$ of cost (or equivalently, size) c_2, we modify in polynomial time this solution to another equal or better solution A' in the following way : in each G_i, if the three triangles covering the edges $x[0]x[1]$, $y[0]y[1]$, and $z[0]z[1]$ (numbered $1, 7, 13$ in Fig. 2) belong to A, we choose the seven *odd* triangles of G_i to belong to A'. If not, we take the six *even* triangles. Let $c_2' \geq c_2$ be the cost of A'. Then, we define a solution $S_f(A)$ of I by choosing the subset c_i to be in $S_f(A)$ if and only if A' contains exactly 7 triangles in G_i. We claim that the pair (f, S_f) is an L-reduction : in each G_i there are 13 different triangles, numbered from 1 to 13 in Fig. 2. The only way to choose 7 edge-disjoint triangles in G_i is by taking all the *odd* triangles, and thus by covering the three edges $x[0]x[1]$, $y[0]y[1]$, and $z[0]z[1]$. All other choices of triangles yield at most 6 edge-disjoint triangles. The key observation is that we are able to choose 7 triangles exactly $OPT(I)$ times. Indeed, each time we choose 7 triangles we cover the edges corresponding to 3 elements of c_i, and since the number of disjoint c_i's in C is $OPT(I)$, this can be done exactly $OPT(I)$ times. On the other hand, one can easily see that $OPT(I) \geq \frac{|C|}{3B}$. Hence :

$$OPT(f(I)) = 7 \cdot OPT(I) + 6(|C| - OPT(I)) \leq OPT(I) + 18B \cdot OPT(I) = (18B+1)OPT(I)$$

To conclude, note that if the solution $S_f(A)$ of I has cost c_1, we have $OPT(I) - c_1 \leq OPT(f(I)) - c_2$. To see this, we observe that $OPT(f(I)) = 6r + OPT(I)$, and also $c_2' = 6r + c_1$, and so $OPT(f(I)) - OPT(I) = c_1 - c_2' \leq c_1 - c_2$.

Both (f, S_f) and (h, S_h) are L-reductions and MAX 3SC-B, $B \geq 3$ and INDEP. SET-B, $B \geq 5$ are MAX SNP-complete [16]. Thus, MECT-B, $B \geq 10$ is MAX SNP-complete.

To prove the last claim, note that the graph $G = (V, E)$ used in the proof is tripartite, where the vertex sets defining the tripartition are :

$$V_0 = \bigcup_{t \in X}^{|X|} t[0] \cup \bigcup_{i=1}^{|C|} \{a_i[2], a_i[5]\}, \quad V_1 = \bigcup_{i=1}^{|C|} \{a_i[j] : j = 1, 4, 7, 8, 9\}, \quad V_2 = \bigcup_{i=1}^{|X|} t[1] \cup \bigcup_{t \in X}^{|C|} \{a_i[3], a_i[6]\} \qquad \square$$

3 APX-Completeness of Traffic Grooming

In this section we prove the hardness results for RING TRAFFIC GROOMING and PATH TRAFFIC GROOMING. First we prove that RING TRAFFIC GROOMING belongs to APX when g is fixed. The same result holds for PATH TRAFFIC GROOMING.

Let us define the *density* ρ of a graph G as its edges-to-vertices ratio : $\rho(G) = \frac{|E(G)|}{|V(G)|}$. To see that RING TRAFFIC GROOMING is in APX for any fixed $g \geq 1$, we have to find a constant-factor approximation algorithm. We use the fact that the best possible density ρ^* of any subgraph used in the partition of the request graph is $\mathcal{O}(\sqrt{g})$, given by (possibly a subgraph of) a circulant graph [4]. We prove

that the cost A of any solution $R_1, \ldots, R_W$ is in the interval $[\frac{|E(R)|}{\rho^*}, 2|E(R)|]$. This clearly implies that any solution has cost at most $2\rho^* = \mathcal{O}(\sqrt{g})$ times the optimal cost. To see this, note that each edge of R contributes at most twice to the cost, so $A \leq 2|E(R)|$. On the other hand, we have

$$A = \sum_{\omega=1}^{W} |V(R_\omega)| = \sum_{\omega=1}^{W} \frac{|E(R_\omega)|}{\rho(R_\omega)} \geq \sum_{\omega=1}^{W} \frac{|E(R_\omega)|}{\rho^*} = \frac{|E(R)|}{\rho^*}$$

Thus, a $\mathcal{O}(\sqrt{g})$-approximation is obtained just by taking *any* partition of the request graph.

Theorem 2. Ring Traffic Grooming *is* APX*-complete for all fixed* $g \geq 1$*. Thus, it does not accept a* PTAS *unless* P = NP*.*

Proof: We prove that Ring Traffic Grooming is APX-complete even if we suppose that the degree of the request graph is bounded by a constant $B \geq 10$. First, we prove the result for $g = 1$. We consider a set of requests R made of a tripartite graph with the three partition classes placed consecutively on the ring, as shown in Fig. 3a. To simplify the presentation, suppose that R can be partitioned into triangles. In any solution, the only possible involved subgraphs are P_2, P_3, P_4, and K_3. It is clear that the best we can do is to groom the requests into triangles (since triangles have the highest density) obtaining an optimal cost of $|E(R)|$. From this we derive that $|E(R)|$ is a lower bound for the number of ADMs of any solution, and that each path used in a given solution adds an additional unity of cost. For each solution S, the additional cost is at least 4/3 times the number of edges covered by paths of S. This bound is tight if all the paths are P_4's. Thus, the number A of ADMs used by S (i.e. the cost of S), satisfies $A \geq (1-\varepsilon)|E(R)| + \varepsilon\frac{4}{3}|E(R)| = (1+\frac{\varepsilon}{3})|E(R)|$, where ε is the percentage of edges of R not covered by triangles in S. By Theorem 2, there exists a constant ε_0 such that we can find in polynomial time at most a fraction $(1-\varepsilon_0)$ of the triangles of R. This means that $(1+\frac{\varepsilon_0}{3})OPT$ is the best solution we may obtain by a polynomial-time algorithm, implying the non-existence of a PTAS.

For $g > 1$, we take a $(2g+1)$-partite graph as the request graph, in such way that each cycle makes at least g tours around the center of the ring. Now, we can reduce the grooming problem to the problem of finding a maximum number of cycles of length $2g+1$ in this graph (as in the case $g = 1$). This later problem is also APX-complete, see Theorem 1. The details can be found in [1]. Hence, Ring Traffic Grooming is Max SNP-complete for bounded number of requests per node $B \geq 10$. □

Theorem 3. Path Traffic Grooming *is* APX*-complete for any fixed* $g \geq 2$*.*

Proof: Again, the result holds even for bounded number B of requests per node, $B \geq 10$. We prove the result for $g = 2$, proceeding for $g > 2$ as in the proof of Theorem 2. Consider a set of requests R made of a tripartite graph with

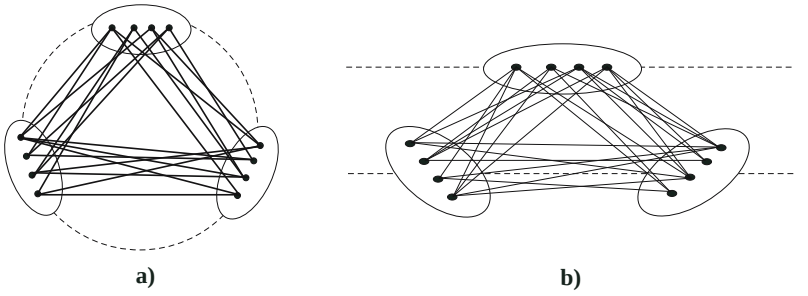


Fig. 3. Request graph used in the proof of APX-completeness of Traffic Grooming : **a)** in the ring for $g = 1$; **b)** in the path for $g = 2$

the three partition classes placed consecutively on the path one after another, as shown in Fig. 3b. Since each triangle induces load 2, minimizing the number of ADMs corresponds to finding the maximum number of edge-disjoint triangles. Therefore, it does not accept a PTAS unless P = NP. □

4 Approximating Ring Traffic Grooming

We are now interested in finding good approximation algorithms considering g as part of the input. As we saw in Sect. 3, obtaining a $\mathcal{O}(\sqrt{g})$-approximation is trivial. Since in practical applications SONET WDM rings are widely used as backbone optical networks [9], [18], the grooming factor is usually greater than the size of the network, i.e. $g \geq n$. Thus, it turns out to be important to find approximation algorithms with an approximation ratio not depending on g. A general approximation algorithm with this property is the main result of this section. It provides in the worst case a $\mathcal{O}(n^{1/3} \log^2 n)$-approximation. We describe it for the ring, but exactly the same arguments provide an algorithm for the path. The main idea is to greedily find subgraphs with high density using approximation algorithms for the Dense k-Subgraph problem, which is defined as follows : given a graph G and an integer k, find an induced subgraph $H \subseteq G$ on k vertices with the highest density among all subgraphs on k vertices. In [11] the authors provide a polynomial-time algorithm with approximation ratio $2n^{1/3}$. To simplify the presentation, suppose that $n = 2^t$ for some $t > 0$:

Algorithm $\mathcal{A}$

Step **1)** Divide the request set into $\log n$ classes, such that in each class C_i the length of the requests lies in the interval $[2^i, 2^{i+1})$, $i = 0, \ldots, \log n - 1$. For each class C_i, the ring can be divided into intervals of length 2^i such that the only requests are between consecutive intervals. In this way we obtain $\frac{n}{2^i}$ subproblems for each class : each one consists in finding an optimal solution in a bipartite graph of size $2 \cdot 2^i$. More precisely, each subproblem can be formulated as :

Bipartite Traffic Grooming
Input : A bipartite graph R, and a grooming factor g.
Output : Partition of the edges of R into subgraphs R_ω with at most g edges, $1 \leq \omega \leq W$.
Objective : Minimize $\sum_{\omega=1}^{W} |V(R_\omega)|$.

Solve all these Bipartite Traffic Grooming subproblems independently, and output the union of all solutions.

Step **2)** To solve each Bipartite Traffic Grooming subproblem in a bipartite graph R, proceed greedily (until all edges are covered) by finding at step i a subgraph R_i of $G \setminus (R_1 \cup \cdots \cup R_{i-1})$ with at most g edges in the following way :
For each $k = 2, \ldots, 2g$ find a subgraph B_k of $R \setminus (R_1 \cup \cdots \cup R_{i-1})$ using the algorithm of [11] for the Dense k-Subgraph problem.

- If for some k^*, $|E(B_{k^*})| > g$, and $|E(B_i)| \leq g$ for all $i < k^*$, remove $|E(B_k^*)| - g$ arbitrary edges of B_{k^*} and replace B_k^* with this new graph. Stop the search at k^*, and output the densest graph among $B_2, \ldots, B_{k^*-1}$, B_{k^*}.
- If not, output the densest subgraph among $B_2, \ldots, B_{2g}$.

Let OPT be the optimal solution of Ring Traffic Grooming, and let OPT_1 be the cost of the solution obtained by solving *optimally* all the subproblems generated by Step **1** of Alg. $\mathcal{A}$. We prove a lemma before stating the theorem.

Lemma 1. *Let β be a given number. Suppose that we can find in any bipartite graph R on at most n vertices, a subgraph with at most g edges which has density at least $1/\beta$ times the density of the densest subgraph with at most g edges. Then in the greedy procedure of Step **2** of Algorithm $\mathcal{A}$ we obtain a solution of cost OPT_2 such that $OPT_2 \leq \mathcal{O}(\log n) \cdot \beta \cdot OPT_1$.*

Proof: Let m be the number of edges of the request graph R, and let $R_1, R_2, \ldots, R_r$ be the subgraphs generated in order by the above algorithm, and covering all the edges. We will prove that $\sum |V(R_i)| \leq \log(m) \cdot \beta \cdot OPT_1$. To prove this, we first enumerate the edges of R in order of appearance in R_i's : all the edges in R_1 will be enumerated $e_1, \ldots, e_{g_1}$ ($g_1 = |E(R_1)| \leq g$), all the edges in R_2 will be enumerated $e_{g_1+1}, \ldots, e_{g_1+g_2}$ ($g_2 = |E(R_2)| \leq g$), and so on. Let ρ_i be the density of the subgraph R_i, i.e. $\rho_i = \frac{|E(R_i)|}{|V(R_i)|}$, and $\Sigma = \sum |V(R_i)|$ the total cost of the solution. For every edge $e_j \in R_i$, we define $c(e_j) = \frac{1}{\rho_i}$. We claim that $\sum_j c(e_j) = \Sigma$. To prove this equality just note that $\sum_{e_j \in E(R_i)} c(e_j) = \frac{|E(R_i)|}{\rho_i} = |V(R_i)|$, and so $\sum_j c(e_j) = \sum_i |V(R_i)| = \Sigma$. Let us define R'_i to be the union of $R_i, R_{i+1}, \ldots, R_r$. We define ρ'_i to be the density of the densest subgraph of R'_i containing at most g edges. Let us take an optimal solution for R'_i, i.e. a decomposition of R'_i into subgraphs $A_1, \ldots, A_s$ such that $\sum_{k=1}^{s} |V(A_k)|$ is minimum. Let $\overline{\rho}_1, \ldots, \overline{\rho}_s$ be the density of these subgraphs. We have :

- $\forall k \leq s, \quad \overline{\rho}_k = dens(A_k) \leq \rho'_i$: because each A_k is a subgraph of R'_i containing at most g edges, and ρ'_i is the density of the densest subgraph with at most g edges in R'_i.
- $\rho'_i \leq \beta\rho_i$: because we suppose that we can find an approximation of ρ'_i up to a factor $1/\beta$.

This implies that $\frac{1}{\overline{\rho}_k} \geq \frac{1}{\beta\rho_i}$, and so $\sum_k |V(A_k)| = \sum_k \frac{|E(A_k)|}{\overline{\rho}_k} \geq \sum_k \frac{|E(A_k)|}{\beta\rho_i} = \frac{|E(R'_i)|}{\beta\rho_i}$

But an optimal solution for R provides a solution for R'_i of cost at least the optimal solution for R'_i, i.e. $\sum_k |V(A_k)| \leq OPT_1$. Using this in the above inequality we get $\frac{1}{\rho_i} \leq \frac{\beta \cdot OPT_1}{|E(R'_i)|}$, and so for an edge $e_j \in R_i$ we have $c(e_j) = \frac{1}{\rho_i} \leq \frac{\beta \cdot OPT_1}{|E(R'_i)|} \leq \frac{\beta \cdot OPT_1}{m-j+1}$, and this proves that

$$\Sigma = \sum_j c(e_j) \leq \beta \cdot (\sum_j \frac{1}{m-j+1}) \cdot OPT_1 \leq \beta \cdot \log(m) \cdot OPT_1 \leq 2\beta \cdot \log(n) \cdot OPT_1$$

□

Theorem 4. *$\mathcal{A}$ is a polynomial-time approximation algorithm that approximates* Ring Traffic Grooming *within a factor $\mathcal{O}(n^{1/3} \log^2 n)$ for any $g \geq 1$.*

Proof: Algorithm $\mathcal{A}$ returns a valid solution of Ring Traffic Grooming, because each request is contained in some bipartite graph, and no request is counted twice. The runtime is polynomial in both n and g, because we run at most $2g-1$ times the algorithm of [11] for each subproblem, and there are $n(\sum_{i=0}^{t-1} \frac{1}{2^i}) - 1 = 2n - 3$ subproblems. We prove the approximation guarantee :

- We claim that $OPT_1 \leq 2 \log n \cdot OPT$. Indeed, let $\overline{c}_i$ be the optimal cost of the subset of requests of length in the interval $[2^i, 2^{i+1})$, $i = 0, \ldots, \log(n)-1$. It is clear that $\overline{c}_i \leq OPT$ for each i , and thus $\sum_{i=0}^{\log n - 1} \overline{c}_i \leq \log n \cdot OPT$. Finally, $OPT_1 \leq 2 \sum_{i=0}^{\log n - 1} \overline{c}_i$, because each vertex is taken into account in two subproblems.
- The greedy procedure described in Step **2** of Algorithm $\mathcal{A}$ outputs a graph whose density is at least $\frac{1}{2n^{1/3}}$ times the highest density (with at most g edges) of the updated request graph. To see that, note that the optimal density is achieved by a subgraph on at most $2g$ vertices (it would be the case of g disjoint edges). Then, for each value of k, the algorithm of [11] finds a $2n^{1/3}$-approximation of the maximum number of edges of an induced subgraph on k vertices[3]. Thus, if we take the densest subgraph among $B_2, \ldots, B_{2g}$ (removing edges if necessary) we also obtain a $2n^{1/3}$-approximation of the highest density of a subgraph with at most g edges. Let ρ_k be the density of B_k before removing edges. The explicit formula of the highest density ρ that we output in Step **2** of Algorithm $\mathcal{A}$ is :

$$\rho := \max_{k \in \{2,\ldots,2g\}} \min\left(\rho_k, \frac{g}{k}\right)$$

[3] In fact, the improved approximation ratio of the Dense k-Subgraph problem is $\mathcal{O}(n^\delta)$ for some constant $\delta < 1/3$ [11]. Obviously, the same applies to our algorithm, replacing the exponent $1/3$ with $\delta < 1/3$.

Looking at the formula we understand why we stop at $k = k^*$ in the algorithm. In other words, we can use $\beta = 2n^{1/3}$ in Lemma 1.

- By combining the remarks above and Lemma 1 we obtain that the cost A returned by Algorithm $\mathcal{A}$ satisfies $A \leq 2n^{1/3} \cdot OPT_2 \leq 4n^{1/3} \log n \cdot OPT_1 \leq 8n^{1/3} \log^2 n \cdot OPT$. □

We can improve the approximation ratio of the algorithm if all the requests have short length compared to the length of the ring. This situation is usual in practical applications since nodes may want to communicate only with their nearest neighbors. Let $f(n)$ be any function of n. If all the requests have length at most $f(n)$, then the above algorithm provides an approximation ratio of $\mathcal{O}(f(n)^{1/3} \log^2 n)$. Indeed, in Step **2** of Algorithm $\mathcal{A}$, we have to find dense subgraphs in bipartite graphs of size at most $2f(n)$, hence the factor $2n^{1/3}$ can be replaced with $2(2f(n))^{1/3}$.

Remark that all the instances of Dense k-Subgraph problem in our algorithm are bipartite. Using the results of [20], it is possible to obtain a better approximation ratio when the request graph is bipartite and satisfies some uniform density conditions. We omit the proof due to lack of space.

Corollary 1. *If the request graph R is such that in any large enough subgraph $H \subseteq R$, a densest subgraph $(A \cup B, E)$ satisfies $|A|, |B| = \mathcal{O}(\sqrt{g})$ and $|E| = \Omega(g)$, then for any constant $\varepsilon > 0$ there exists a polynomial-time algorithm for* Ring Traffic Grooming *with approximation ratio $\mathcal{O}(n^{\varepsilon} \log^2 n)$.*

To end this section, note that the results of [7] show that the density can be approximated within a constant factor two in the class of graphs excluding a fixed graph H as minor. Thus, if the request graph R is H-minor free (for instance if R is planar, or of bounded genus,...), our algorithm achieves an approximation factor of $\mathcal{O}(\log^2 n)$.

5 Conclusions and Further Research

This article deals with Traffic Grooming, a central problem in WDM optical networks. The contribution of this work can be divided in two main parts : on the one hand, we state the first hardness results for Ring Traffic Grooming and Path Traffic Grooming for fixed values of g. More precisely, we prove that Ring Traffic Grooming is APX-complete for fixed $g \geq 1$, and that Path Traffic Grooming is APX-complete for fixed $g \geq 2$. In other works, we rule out the existence of a PTAS for fixed values of g. To prove this results we reduce Ring Traffic Grooming for $g = 1$ to the problem of finding the maximum number of edge-disjoint triangles in a graph of degree bounded by B (MECT-B for short). We prove that MECT-B is APX-complete, and we generalize this reduction for Path Traffic Grooming and for all values of $g \geq 1$. On the other hand, we provide the first polynomial-time approximation algorithm for Ring and Path Traffic Grooming with an approximation ratio not depending on g, considering g as part of the input.

There remains still a lot of work to be done. It is a challenging open problem to close the complexity gap of Traffic Grooming, that is, to provide an approximation algorithm with an approximation ratio matching the corresponding inapproximability result. We are convinced that the inherent difficulty of the problem resides in finding dense subgraphs with bounded number of edges. This problem is strongly related to the problem of finding the densest subgraph with bounded number of vertices, which has been recently proved to have, essentially, the same difficulty as the Dense k-Subgraph problem [2]. The non-existence of a PTAS for the Dense k-Subgraph problem has been proved in [17] involving very technical proofs, and this is the best existing hardness result. A long-standing conjecture claims that there exists some constant $\varepsilon > 0$ such that finding a n^{ε}-approximation for Dense k-Subgraph is NP-hard [11]. As we proved in Sect. 4, an α-approximation for Dense k-Subgraph yields a $\mathcal{O}(\alpha \log^2 n)$-approximation for Ring Traffic Grooming. We suspect that a similar result in the other direction should also exist. Because of this, we conjecture that :

Conjecture 1. There exists some constant $\delta > 0$, such that Ring Traffic Grooming is hard to approximate within n^{δ} when g is part of the input.

Acknowledgement. Many thanks to David Coudert, Mordechai Shalom and Shmuel Zaks for helpful discussions.

References

1. Amini, O., Pérennes, S., Sau, I. : Hardness and Approximation of Traffic Grooming. Research Report 6236, INRIA, Accessible from 3rd author's homepage (2007)
2. Andersen, R. : Finding large and small dense subgraphs. Submitted for publication (arXiv :cs/0702032v1) (February 2007)
3. Bermond, J.-C., Coudert, D.
4. Bermond, J.-C., Coudert, D., Muñoz, X., Sau, I. : Traffic Grooming in Bidirectional WDM Ring Networks. In : IEEE-LEOS ICTON / COST 293 GRAAL, vol. 3, pp. 19–22 (2006)
5. Chiu, A., Modiano, E. : Traffic grooming algorithms for reducing electronic multiplexing costs in WDM ring networks. IEEE/OSA Journal of Lightwave Technology 18(1), 2–12 (2000)
6. Chow, T., Lin, P. : The ring grooming problem. Networks 44, 194–202 (2004)
7. Demaine, E., Hajiaghayi, M.T., Kawarabayashi, K.C. : Algorithmic graph minor theory : Decomposition, approximation and coloring. In : 46th Annual IEEE Symposium on Fondations of Computer Science (FOCS), pp. 637–646 (2005)
8. Dutta, R., Rouskas, G. : On Optimal Traffic Grooming in WDM Rings. In : Proceedings of ACM Sigmetrics/Performance, pp. 164–174, Cambridge (2001)
9. Dutta, R., Rouskas, N. : Traffic grooming in WDM networks : Past and future. IEEE Network 16(6), 46–56 (2002)
10. Epstein, L., Levin, A. : Better Bounds for Minimizing SONET ADMs. In : Persiano, G., Solis-Oba, R. (eds.) WAOA 2004. LNCS, vol. 3351, pp. 281–294. Springer, Heidelberg (2005)
11. Feige, U., Peleg, D., Kortsarz, G. : The Dense k-Subgraph Problem. Algorithmica 29(3), 410–421 (2001)

12. Flammini, M., Moscardelli, L., Shalom, M., Zaks, S. : Approximating the traffic grooming problem. In : Deng, X., Du, D.-Z. (eds.) ISAAC 2005. LNCS, vol. 3827, pp. 915–924. Springer, Heidelberg (2005)
13. Gerstel, O., Ramaswami, R., Sasaki, G. : Cost Effective Traffic Grooming in WDM Rings. In : Proceedings of INFOCOM, San Francisco, pp. 69–77 (1998)
14. Holyer, I. : The NP-Completeness of Some Edge-Partition Problems. SIAM Journal on Computing 10(4), 713–717 (1981)
15. Huang, S., Dutta, R., Rouskas, G.N. : Traffic Grooming in Path, Star, and Tree Networks : Complexity, Bounds, and Algorithms. IEEE Journal on Selected Areas in Communications 24(4), 66–82 (2006)
16. Kann, V. : Maximum bounded 3-dimensional matching is MAX SNP-complete. Information Processing Letters 37, 27–35 (1991)
17. Khot, S. : Ruling out PTAS for graph min-bisection, densest subgraph and bipartite clique. In : FOCS, pp. 136–145 (2004)
18. Modiano, E., Lin, P. : Traffic grooming in WDM networks. IEEE Communications Magazine 39(7), 124–129 (2001)
19. Shalom, M., Unger, W., Zaks, S. : On the Complexity of the Traffic Grooming Problem in Optical Networks
20. Suzuki, A., Tokuyama, T. : Dense subgraph problem revisited. In : Joint Workshop "New Horizons in Computing" and "Statistical Mechanical Approach to Probabilistic Information Processing", Japan (July 2005)
21. Zhu, K., Mukherjee, B. : A review of traffic grooming in WDM optical networks : Architectures and challenges. Optical Networks 4(2), 55–64 (2003)

Depth of Field and Cautious-Greedy Routing in Social Networks*

David Barbella[1], George Kachergis[1,2], David Liben-Nowell[1], Anna Sallstrom[1], and Ben Sowell[1,3]

[1] Department of Computer Science, Carleton College, Northfield, MN 55057, USA
[2] Dept. of Psych. & Brain Sciences, Indiana University, Bloomington, IN 47405, USA
[3] Department of Computer Science, Cornell University, Ithaca, NY 14853, USA
{barbelld,dlibenno,sallstra}@carleton.edu, gkacherg@indiana.edu, sowell@cs.cornell.edu

Abstract. Social networks support efficient decentralized search: people can collectively construct short paths to a specified target in the network. *Rank-based friendship*—where the probability that person u befriends person v is inversely proportional to the number of people who are closer to u than v is—is an empirically validated model of acquaintanceship that provably results in efficient decentralized search via greedy routing, even in networks with variable population densities. In this paper, we introduce *cautious-greedy routing*, a variant of greedy that avoids taking large jumps unless they make substantial progress towards the target. Our main result is that cautious-greedy routing finds a path of short expected length from an arbitrary source to a randomly chosen target, independent of the population densities. To quantify the expected length of the path, we define the *depth of field* of a metric space, a new quantity that intuitively measures the "width" of directions that leave a point in the space. Our main result is that cautious-greedy routing finds a path of expected length $O(\log^2 n)$ in n-person networks that have aspect ratio polynomial in n, bounded doubling dimension, and bounded depth of field. Specifically, in k-dimensional grids under Manhattan distance with arbitrary population densities, the $O(\log^2 n)$ expected path length that we achieve with the cautious-greedy algorithm improves the best previous bound of $O(\log^3 n)$ with greedy routing.

1 Introduction

As large-scale datasets of social interactions have become widespread, computer scientists have begun to explore *social networks*, graphs in which nodes representing people are connected by edges representing friendships. Social networks are a particularly appealing domain for interdisciplinary application of computational thinking: a graph-theoretic, algorithmic approach can lend interesting insight

* Supported in part by grants from Carleton College and by NSF grant CCF-0728779. Thanks to Esteban Arcaute, Seth Gilbert, Ravi Kumar, Jeff Ondich, Andrew Tomkins, and Sergei Vassilvitskii for helpful conversations.

T. Tokuyama (Ed.): ISAAC 2007, LNCS 4835, pp. 574–586, 2007.

to questions traditionally in the domain of the social sciences. The well-known *small-world phenomenon*—first observed explicitly by Stanley Milgram's ingenious experiments [21], in which people demonstrated an ability to collectively construct short chains of acquaintances to pass a message to a specified target person—is a marked example [15]. A key observation of Jon Kleinberg [12, 13], made some thirty years after Milgram's original experiment, is that these results are at heart algorithmic: people are able to use some sort of distributed algorithm to construct short paths through the social network, with each node having only limited, local information about the friendships in the network. Kleinberg gave a simple model that suffices to produce a navigable small world. Start from a k-dimensional grid of people, with $k = \Theta(1)$, where we view proximity in the grid as corresponding to similarity in geographic location, occupation, or some other attribute. Connect each person to $2k$ *local neighbors*, her immediate neighbors in the grid. Also endow each person u with one *long-range link*, chosen randomly so that $\mathtt{Pr}[u \to v] \propto d(u,v)^{-\beta}$, where $d(u,v)$ is the Manhattan distance between u and v and $\beta \geq 0$ is a parameter of the model. (The presence of additional long-range links does not substantially affect the results.) Kleinberg studied *greedy routing*—to route a message from s to t, person s forwards the message to the friend of s who is closest in lattice distance to t—and showed that this algorithm finds paths of expected length $O(\log^2 n)$ in n-person networks when $\beta = k$, and of length $n^{\Omega(1)}$ when $\beta \neq k$. A $\Omega(\log^2 n)$ bound on the expected length of the path found by greedy routing when $\beta = k$ has also been shown [4, 20].

These results have subsequently been extended and adapted to situations in which the underlying network is not a grid, but is instead, for example, a tree [14] or a network that has low treewidth [8], bounded growth rate [6, 7], or low doubling dimension [24]. But an important feature present in the real world is lacking in almost all of these models: whatever one chooses as the space in which to measure similarity, people are not uniformly distributed among the points in this space. The *group-structure* model [14] can handle differential population density, as can *rank-based friendship* [16, 18]. We focus on the latter, which has been shown empirically to be a close match for friendship patterns in a real large-scale online social network [18]. The model is still based on an underlying similarity measure, but an arbitrary number of people can live at each point. Define the *rank* of a person v with respect to u as the number of people who live at least as close to u as v does. We generate long-range links probabilistically using rank instead of distance: a person u chooses a long-range link v so that $\mathtt{Pr}[u \to v]$ is proportional to the rank of v with respect to u. (Rank-based friendship implicitly handles the dimensionality of the underlying similarity measure: in a k-dimensional grid with $\Theta(1)$ population at each point, we have $\mathsf{rank}_u(v) = \Theta(d(u,v)^k)$, matching the navigable distribution that explicitly involves k in Kleinberg's distance-based setting.)

In joint work of Ravi Kumar, Andrew Tomkins, and the third author [16], greedy routing was shown to perform well in rank-based social networks, in the following sense. In a $\Theta(1)$-dimensional grid with arbitrary positive population at every point and with total population n, the greedy algorithm finds a path

of expected length $O(\log^3 n)$ from an arbitrary source to the point of a target chosen uniformly from the population. In fact, these results were shown in a much more general setting [16]: if similarity is measured by proximity in any metric space, then GREEDY finds a path of expected length $O(\log n \cdot \log^2 \Delta \cdot 2^{O(\alpha)})$ to a randomly chosen target, where n is the population size; Δ is the distance between maximally distant people; and α is the doubling dimension of the metric space, a combinatorial measure of its implicit dimensionality (see [10,2,24], e.g.).

Our contributions. The theorem for rank-based networks with variable population densities restricted to k-dimension grids under Manhattan distance says that GREEDY finds a path of expected length $O(\log^3 n)$ to a randomly chosen target. This result is weaker than Kleinberg's theorem in two ways: there is an additional $O(\log n)$ factor in the path length; and there is "in expectation for a random target" in place of "for any target." (These negatives are, of course, counterbalanced by the increased generality of the model, which can handle essentially arbitrary population distributions.)

In this paper, we improve the upper bound on expected path length to $O(\log^2 n)$ in k-dimensional grids with arbitrary population densities, closing the gap with Kleinberg's analysis of the uniform-population case. We achieve this bound with the *cautious-greedy algorithm*, a variation on greedy routing that we introduce. Intuitively, the greedy algorithm can get into trouble in variable-density networks as follows. Imagine a person s who only lives near points with unit population. Suppose that s has a friend u so that $d(u,t) = d(s,t) - \varepsilon$ but the jump from s to u "overshoots" the target t. If there is a city with massive population near u (but farther from t than u is), then s may be making a mistake by choosing u as the next step in the chain: because u is in the "shadow" of the city, then we have $\mathtt{Pr}[s \to t] \gg \mathtt{Pr}[u \to t]$. Indeed, the same is true from u for all points close to t, and showing that GREEDY is making progress at every step is difficult. (This difficulty seems intuitive: if s and t live 60 and 25 miles due west of New York City, respectively, then overshooting t from s by jumping to a friend in Manhattan seems like a bad idea, because the chain is now stuck inside New York's "basin of attraction.") CAUTIOUSGREEDY differs from GREEDY in that it conservatively takes local links *unless* there is a long-range link that halves the distance to the target, in which case it follows that link. We are able to show that, in rank-based social networks with arbitrary population distributions derived from k-dimensional grids under Manhattan distance, the expected length of the CAUTIOUSGREEDY(s,t) path for a randomly chosen target t is $O(\log^2 n)$. We also show that this bound is tight in the 1-dimensional case.

The result for k-dimensional grids is a corollary of our main theorem, which shows that cautious-greedy routing performs well in expectation for social networks with similarity measures derived from arbitrary metric spaces. As usual, the performance of the algorithm depends on certain properties of the metric space, including the aspect ratio Δ and the doubling dimension α. We also uncover a new quantity $\rho \in (0,1]$ characterizing metric spaces, intuitively measuring the smallest "width" of each direction leaving a point in the space, that is crucial to our analysis. By analogy to the corresponding concept in photography,

we call ρ the *depth of field* of the metric space. More precisely, the depth of field is the minimum over points s and t of the ratio $r(s,t)/d(s,t)$, where $r(s,t)$ is the maximum radius of a ball B around t so that a shortest path from s to every point in B has the same first step. In networks with large depth of field (e.g., Manhattan distance in k-dimensional grids), we can prove much better routing bounds than in networks with small depth of field (e.g., Euclidean distance in k-dimensional grids). Our main result is that CautiousGreedy finds a path of expected length $O(\log n \cdot \log \Delta \cdot (c\rho)^{-\alpha})$ for constant c, improving by a $O(\log \Delta)$ factor the bound from [16] in networks with bounded ρ and α.

Other related work. Although Greedy is the most commonly analyzed decentralized social-network routing algorithm, there has been significant work on other algorithms as well. Typically—and in contrast to CautiousGreedy, which still uses only completely local information in constructing the path—these algorithms endow individuals with additional structural information about the network, such as awareness of friend's friends (e.g., [9,19,20,17,24]). Other studies of the navigability of social networks and, in particular, good local-information algorithms for navigation have been performed, largely focusing on simulations and empirical studies of real networks (e.g., [23, 25, 3, 5]). There has also been recent relevant work in metric embeddings in which the underlying metric space can be simplified without large distortion of distances between nodes that are close together [1] and in designing peer-to-peer systems where node distributions are nonuniform in keyspace [11].

2 Depth of Field in Metric Spaces

Consider a metric space $\mathcal{M} = \langle X, d \rangle$. For convenience, throughout the paper we scale every metric space so that $\min_{x \neq y} d(x,y) = 1$. We first mention a few standard notions that we use throughout:

- Let $\Delta := \max_{x,y \in X} d(x,y)$ denote the *aspect ratio* of the metric space.
- Let $B_r(x) := \{y : d(x,y) < r\}$ denote the open radius-r ball around $x \in X$.
- Let α denote the *doubling dimension* of the metric space: α is the smallest value such that, for every $r > 0$, every set $Y \subseteq X$ of radius $2r$ can be covered by at most 2^α subsets of X, each of radius r.

We will need to develop a notion that quantifies the following intuition: if a point s is far away from a point t, then making a small step closer to t from s should bring one closer to points near t as well. We will define the *depth of field* of the metric space to quantify the notion of "near t." (Our use of this term is inspired by the same term in photography: for a camera focused on a point t, "depth of field" refers to the range of points around t that are also in focus. A large depth of field means that many points near t are also in focus.) Let $r > 0$ denote the largest radius so that some point u is on a shortest path from s to *every* point in the ball $B_r(t)$. The farther apart s and t are, the larger one would expect r to be; thus, we will be concerned with the ratio between r and $d(s,t)$.

Definition 1 (Depth of field). *For arbitrary points $s, t, u \in X$ with $s \notin \{t, u\}$, define $r_u(s,t)$ as the maximum value such that*

$$\forall z \in X \qquad d(z,t) < r_u(s,t) \Rightarrow d(s,z) = d(s,u) + d(u,z).$$

We define the depth of field *of s and t as $\rho(s,t) := \max_{u \neq s} r_u(s,t)/d(s,t)$. The* depth of field *of the metric space $\mathcal{M}$ is $\rho(\mathcal{M}) := \min_{s \neq t} \rho(s,t)$.*

Lemma 2. *For any metric space $\mathcal{M}$, we have $0 < \rho(\mathcal{M}) \leq 1$.*

Proof. Consider arbitrary distinct s, t. For sufficiently small r, the ball $B_r(t)$ is just $\{t\}$. So $r_t(s,t) > 0$, and $\rho(s,t) > 0$. On the other hand, we have $s \in B_r(t)$ for $r > d(s,t)$. Thus $r_u(s,t) \leq d(s,t)$ for every u. Thus $0 < \rho(s,t) \leq 1$. □

One can give similar intuitive descriptions of the depth of field and the doubling dimension of a metric space: they both aim to quantify the number of distinct directions emanating from a point in the metric space. However, they measure "number of distinct directions" in different ways. Informally, doubling dimension counts something like the number of directions one can go from s; depth of field counts something like the "width" of the narrowest of these directions.

Denote depth of field and doubling dimension by ρ and α, respectively. In what follows, we will find algorithms whose running times depend exponentially on α and $1/\rho$, so we are most interested in metric spaces where both α and $1/\rho$ are bounded. To clarify the relationship between these quantities, we note metric spaces where one or the other or both of these quantities is/are small:

- a k-dimensional grid under Manhattan distance has $\rho = 1/k$ and $\alpha = \Theta(k)$.
- a 2-by-n grid under Euclidean distance has $\rho = 1/n$ and $\alpha = \Theta(1)$.
- a metric space in which distances are given by shortest paths in an n-node star graph (a tree with $n-1$ leaves) has $\rho = 1$ and $\alpha = \log n$.

3 Social Networks, Rank-Based Friendship, and Routing Algorithms

A *social network* is a directed graph $\langle P, E \rangle$, where a node represents a person and an edge denotes a friendship between its endpoints. Let $\Gamma(u)$ denote the friends of $u \in P$. Our general framework consists of some attributes of the people in P, with friendships derived in some way from these attributes. (We use terminology suggestive of a geographic interpretation, but other attributes are equally valid in this setting.) The attributes will be globally known to all people, but the friendships will be known only to the people involved. As a formal encoding of this framework, we consider the following structures throughout:

- a finite set L of *points*;
- a *distance metric* $d : L \times L \to \mathbb{R}^{\geq 0}$ on the points (so $\langle L, d \rangle$ is a metric space);
- a finite set P of *people*; and
- a *location function* $\mathsf{loc} : P \to L$ so that $\mathsf{loc}(u)$ is the point where $u \in P$ lives.

We will permit ourselves to write an element u of P in contexts where an element of L is expected, with the understanding that u is shorthand for $\mathsf{loc}(u)$. For $\ell \in L$ or $L' \subseteq L$, let $\mathsf{pop}(\ell) := |\{u \in P : \mathsf{loc}(u) = \ell\}|$ and $\mathsf{pop}(L') := \sum_{\ell \in L'} \mathsf{pop}(\ell)$ denote the *population* of a point or set of points. Write $n := \mathsf{pop}(L) = |P|$ for the total population. We will also impose a condition called *neighbor connectivity* [16] on the social networks that we consider. For every $p, q \in P$ with $\mathsf{loc}(q) \neq \mathsf{loc}(p)$, we require that p have a friend in some location ℓ such that $d(\mathsf{loc}(p), \ell) + d(\ell, \mathsf{loc}(q)) = d(\mathsf{loc}(p), \mathsf{loc}(q))$. (Formally, if G_d is the minimal graph on L where shortest paths correspond to the metric d, then p must have a friend in every neighbor of $\mathsf{loc}(p)$ in G_d.) Among other things, this guarantees that, for people $s, t \in P$ with $\mathsf{loc}(s) \neq \mathsf{loc}(t)$, person s has a friend u with $d(s,t) > d(u,t)$.

3.1 Rank-Based Friendships

For two people $u, v \in P$, the *rank of v with respect to u* is the number of people $w \in P$ who are closer to u than v, so $\mathsf{rank}_u(v) := |\{w \in P : d(u,w) < d(u,v)\}|$. For concreteness, we will specify that $\mathsf{rank}_u(u) := 1$ for every person $u \in P$. Ties in distance are broken using a canonical ordering on P, so for any person $u \in P$ and any rank $i \in \{1, \ldots, n\}$, there is exactly one person v such that $\mathsf{rank}_u(v) = i$. A *rank-based friendship* for a person $u \in P$ is one generated as follows: a friend v is chosen randomly for u according to the probability distribution $\Pr[u \text{ links to } v] \propto 1/\mathsf{rank}_u(v)$. Let $H_n = \Theta(\log n)$ denote the nth harmonic number. By normalization, we have the following:

$$\Pr[\text{a rank-based link from } u \text{ is } u \to v] = 1/(H_n \cdot \mathsf{rank}_u(v)). \tag{1}$$

We endow each person with ≥ 1 rank-based friendship, chosen according to (1), along with an arbitrary set of *local neighbors* satisfying neighbor connectivity.

3.2 Decentralized Routing Algorithms

Given *source* and *target* individuals $s, t \in P$, a *routing algorithm* seeks a path $\sigma = \langle u_0, u_1, \ldots, u_k \rangle$ from $s = u_0$ to u_k with $\mathsf{loc}(u_k) = \mathsf{loc}(t)$ in the graph $\langle P, E \rangle$. (We do not model routing within points in this paper.) A *decentralized* algorithm computes the next step u_{i+1} from the current person u_i without taking the entire graph $\langle P, E \rangle$ as input: only the edges in E incident to u_i are available to the algorithm. (Full information about L, d, P, and loc is available to the algorithm; thus, for example, a person s can compute $r_u(s,t)$ for any u, t in the sense of Def. 1.) We focus on two particular decentralized algorithms. Under the well-studied *greedy algorithm*, the current person u_i simply chooses her friend who is closest to t as the next step in the path. We also introduce and analyze the *cautious-greedy algorithm*, which is a conservative variant of Greedy that refuses to take long jumps that do not make significant progress (specifically, by halving the distance) to the target, instead opting for a "safe" local link.

CautiousGreedy(s,t):

(halving step.) Let $u := \operatorname{argmin}_{u \in \Gamma(s)} d(u,t)$. If $d(u,t) \leq \frac{d(s,t)}{2}$, forward to u.
(nonhalving step.) Else forward to $\operatorname{argmax}_{u \in \Gamma(s)} r_u(s,t)$ in the sense of Def. 1.

Lemma 3. *Suppose the network is neighbor connected and let ρ be the depth of field of $\langle L, d\rangle$. If* CautiousGreedy(s,t) *takes a nonhalving step from s to w, then $d(s,z) = d(s,w) + d(w,z)$ for every z such that $d(z,t) < \rho \cdot d(s,t)$.* □

(The proof is omitted due to space constraints.) As a corollary, we note that both Greedy and CautiousGreedy make strict progress towards their targets in every step. Among other things, this fact guarantees that every person is encountered only once by a run of the algorithm. Thus we can invoke the principle of deferred decisions in our analysis (see [22]): we proceed as if the long-range links of each person are generated only once the algorithm encounters that person.

4 An Upper Bound for Cautious Greedy Routing

Consider fixed L, d, P, loc as defined previously. Let ρ and α, respectively, denote the depth of field and doubling dimension of the metric space $\langle L, d\rangle$. We will be interested in the length of the path found by CautiousGreedy(s,t), where $s \in P$ is arbitrary and $t \in P$ is chosen uniformly at random from the population. Most of our effort will be focused on analyzing the number of steps required to halve the distance to the target—or, more precisely, to reach a person u such that $d(u,t) < 2^{i-1}$ from a source s when t is chosen randomly from $\{t : d(s,t) < 2^i\}$.

Halving the Distance to the Target

Fix a source person s and some integer $i \in \mathbb{Z}^{\geq 1}$. Let $B := B_{2^i}(s)$ denote the ball of radius 2^i around s. It will turn out to be handy to fix notation for $D := B_{2^i}(s) - \{z : d(s,z) \leq 2^{i-1}\}$, the "donut" formed by removing the (closed) ball of radius 2^{i-1} centered at s from B.

We will use a cover of B by a small (in terms of α and $1/\rho$) set of balls of radius $\rho \cdot 2^{i-2}$. This radius is chosen carefully to ensure the following: whenever we take a nonhalving step towards target t from a node u with $d(u,t) \geq 2^{i-1}$, we have stepped along a shortest path to every node in the ball containing t that is included in the cover.

Lemma 4. *There is a set $\mathscr{Q} \subseteq B$ where $|\mathscr{Q}| \leq (8/\rho)^\alpha$ and $\bigcup_{\ell \in \mathscr{Q}} B_{\rho \cdot 2^{i-2}}(\ell) \supseteq B$.*

Proof (sketch). By repeatedly applying the definition of doubling dimension, we find that we can cover B with a set of at most $2^{(3+\log_2(1/\rho))\alpha} = (8/\rho)^\alpha$ balls of radius at most $\rho \cdot 2^{i-2}$ that cover B. We let $\mathscr{Q}$ denote the set of their centers. □

Throughout this section, fix $\mathscr{Q}$ to denote this set, so that, for every $u \in B$, there exists a point $q \in \mathscr{Q}$ such that $d(u,q) \leq \rho \cdot 2^{i-2}$. Write $q(u) := \operatorname{argmin}_{q \in \mathscr{Q}} d(q,u)$, and write $C(u) := B \cap B_{\rho \cdot 2^{i-2}}(q(u))$. Notice that $u \in C(u)$.

Lemma 5. *Let $t \in B$ be arbitrary. Suppose* CautiousGreedy(s,t) *arrives at node u with $d(u,t) > 2^{i-1}$. Let $v \in C(t)$ be arbitrary. Then $B_{d(u,v)}(u) \subseteq B$.*

Proof (by induction on the number of steps taken by CautiousGreedy(s,t)*).* If no steps have been taken by CautiousGreedy(s,t), then the claim is trivial, because $u = s$ and $C(t) \subseteq B$. Otherwise, suppose that u was reached by a step from a node w. Notice that every step that CautiousGreedy(s,t) has taken before reaching u must be nonhalving: otherwise, there was a halving step from some u^* with $d(s,t) \geq d(u^*,t)$ to a neighbor v^* with $d(v^*,t) \leq d(u^*,t)/2$; but then u would not satisfy the conditions of the lemma, as $d(v^*,t) \leq d(u^*,t)/2 \leq d(s,t)/2 \leq 2^{i-1}$. Observe that $d(v,t) < \rho \cdot 2^{i-1}$:

$$d(v,t) \leq d(v,q(t)) + d(q(t),t) < (\rho \cdot 2^{i-2}) + (\rho \cdot 2^{i-2}) = \rho \cdot 2^{i-1}.$$

by the triangle inequality and the fact that $v, t \in C(t) \subseteq B_{\rho \cdot 2^{i-2}}(q(t))$ by the definition of v and $C(t)$. We also have that $d(w,t) > d(u,t) > 2^{i-1}$ by assumption. Therefore, because we took a nonhalving step from w to u, Lemma 3 implies that $d(w,v) = d(w,u) + d(u,v)$, because $d(v,t) < \rho \cdot 2^{i-1}$ implies $d(v,t) < \rho \cdot d(w,t)$. To complete the proof of the lemma, consider a generic point x. We show that $x \in B_{d(u,v)}(u) \Rightarrow x \in B$:

$$\begin{aligned}
x \in B_{d(u,v)}(u) &\iff d(x,u) < d(u,v) \\
&\iff d(x,u) < d(w,v) - d(w,u) \\
&\qquad\qquad (d(w,v) = d(w,u) + d(u,v) \text{ as above}) \\
&\Rightarrow d(x,w) < d(w,v) \\
&\qquad (d(x,w) \leq d(x,u) + d(u,w) \text{ by the triangle inequality}) \\
&\iff x \in B_{d(w,v)}(w) \\
&\Rightarrow x \in B \qquad\qquad \text{(inductive hypothesis).} \quad \square
\end{aligned}$$

For arbitrary $t \in B$, let X_t be a random variable denoting the number of steps that CautiousGreedy(s,t) takes before it reaches a person u with $d(u,t) \leq 2^{i-1}$. Note that, for $t \in B - D$, we have $X_t = 0$: the node s itself satisfies the desired condition for u. For $t \in D$, it suffices to reach a node $u \in C(t)$: we have

$$d(u,t) \leq d(u,q(t)) + d(q(t),t) < \rho \cdot 2^{i-2} + \rho \cdot 2^{i-2} = \rho \cdot 2^{i-1} \leq 2^{i-1}$$

by the triangle inequality, the definition of $C(t)$ and $q(t)$ and the fact that $t \in C(t)$, and Lemma 2. Note CautiousGreedy(s,t) will take a halving step to follow a link to any node in $C(t)$ if $d(s,t) > 2^{i-1}$, as $d(u,t) < 2^{i-2} < d(s,t)/2$; thus it suffices to compute the probability that a node has a link to $C(t)$.

Lemma 6. *For $t \in D$, we have $\mathtt{E}[X_t] \leq H_n \cdot \mathsf{pop}(B)/\mathsf{pop}(C(t))$.*

Proof. Suppose CautiousGreedy has generated a partial path to t and that u is the last node on that path. If $d(u,t) \leq 2^{i-1}$, we are done, so consider a node u such that $2^{i-1} < d(u,t) \leq d(s,t) \leq 2^i$, as every step of CautiousGreedy moves closer to the target t. The probability that a rank-based link from such a node u goes into $C(t)$ is

$$\mathtt{Pr}[u \to C(t)] = \sum_{v \in C(t)} \frac{1}{H_n \cdot \mathsf{rank}_u(v)} \geq \sum_{v \in C(t)} \frac{1}{H_n \cdot \mathsf{pop}(B)} = \frac{\mathsf{pop}(C(t))}{H_n \cdot \mathsf{pop}(B)} \quad (2)$$

by (1) and Lemma 5. Thus at every step while CautiousGreedy is farther than 2^{i-1} from the target, the probability of the current node having a rank-based link to $C(t)$ is given by (2). Thus the expected number of steps until CautiousGreedy either reaches a point within distance 2^{i-1} of t through non-halving steps or reaches a person in $C(t)$ via a halving step is no larger than the expected waiting time for success in geometric random process with success probability $\mathsf{pop}(C(t))/(H_n \cdot \mathsf{pop}(B))$. The claim follows. □

Lemma 7. *Let t be a target chosen uniformly at random from B. Then the expected length of* CautiousGreedy(s,t) *before it arrives at some node u with $d(u,t) \le 2^{i-1}$ is at most $|\mathscr{Q}| \cdot H_n$, where the expectation is taken both over the random construction of the network and over the random choice of t.*

Proof. Let X be a random variable denoting the length of the path found by CautiousGreedy(s,t) before it arrives at a node u with $d(u,t) \le 2^{i-1}$ when t is drawn uniformly at random from B. Then

$$\begin{aligned}
\mathtt{E}[X] &= \mathtt{E}[X \mid t \in D] \cdot \mathtt{Pr}[t \in D \mid t \in B] \\
&\quad + \mathtt{E}[X \mid t \in B - D] \cdot \mathtt{Pr}[t \in B - D \mid t \in B] \\
&= \mathtt{E}[X \mid t \in D] \cdot \mathtt{Pr}[t \in D \mid t \in B] \qquad (t \in B - D \text{ is done in zero steps}) \\
&= \left[\sum_{t \in D} \mathtt{E}[X_t] \cdot \frac{1}{\mathsf{pop}(D)}\right] \cdot \frac{\mathsf{pop}(D)}{\mathsf{pop}(B)} \\
&\le \sum_{t \in D} \frac{H_n \cdot \mathsf{pop}(B)}{\mathsf{pop}(C(t))} \cdot \frac{1}{\mathsf{pop}(D)} \cdot \frac{\mathsf{pop}(D)}{\mathsf{pop}(B)} \qquad \text{(Lemma 6)} \\
&= H_n \sum_{t \in D} \frac{1}{\mathsf{pop}(C(t))} \\
&= H_n \sum_{q \in \mathscr{Q}} \frac{\mathsf{pop}(C(q))}{\mathsf{pop}(C(q))} \quad \left(\tfrac{1}{\mathsf{pop}(C(q))} \text{ is summed once for each person in } C(q)\right) \\
&= H_n \cdot |\mathscr{Q}|. \qquad \square
\end{aligned}$$

Reaching the Target

Lemma 7 establishes that, for a fixed source s and a fixed i, the expected number of steps for CautiousGreedy to get to within distance 2^{i-1} of a target chosen uniformly at random from the ball of radius 2^i around s is small as long as $|\mathscr{Q}|$ is small. By repeated invocations of this lemma, we can establish a polylogarithmic upper bound on the expected length of the path found by CautiousGreedy:

Theorem 8. *Fix a neighbor-connected rank-based social network with population size n, depth of field ρ, doubling dimension α, and aspect ratio Δ. Let s be an arbitrary source person, and let t be a target person chosen uniformly at random from the population. Then the expected length of the* CautiousGreedy(s,t) *path from s to $\mathsf{loc}(t)$ is $O(\log n \cdot \log \Delta \cdot (8/\rho)^\alpha)$, where the expectation is taken over both the random construction of the network and the random choice of t.*

Proof. Let $Y_{u,i}$ be a random variable that denotes the number of people that CautiousGreedy(u,t) encounters before it reaches a person within distance 2^{i-1} of a target t chosen uniformly at random from $B_{2^i}(u)$. For any u and i, by Lemma 7, we have $\mathtt{E}[Y_{u,i}] \leq |\mathscr{Q}| \cdot H_n \leq (8/\rho)^{\alpha} \cdot H_n$. Choose a target t uniformly at random from the population. For any u and any i, conditioned on $d(u,t) < 2^i$, the target t is a uniformly chosen person from $B_{2^i}(u)$. Thus, starting from any source node u_i, conditioned on the distance to the target being at most 2^i, the expected number of steps before CautiousGreedy reaches a node within distance 2^{i-1} of the target is $O(\log n \cdot (8/\rho)^{\alpha})$. The total length of the CautiousGreedy(s,t) is at most the number of steps required to reduce the distance to the target from Δ down to $1/2$—i.e., $O(\log \Delta)$ iterations of this process. By linearity of expectation and the above bound, the theorem follows. □

5 A Tight Lower Bound on Cautious Greedy

We have shown that $\mathtt{E}_t[|\text{CautiousGreedy}(s,t)|] = O(\log^2 n)$ in networks with constant doubling dimension, constant depth of field, and aspect ratio that is polynomial in the population size. In this section, we exhibit a network with $\alpha = \Theta(1)$, $1/\rho = \Theta(1)$, and $\Delta = n$ such that the expected length of the path CautiousGreedy(s,t) for a randomly chosen t is $\Omega(\log^2 n)$. Our results rely heavily on a lower bound proven by Martel and Nguyen [20] on Greedy: in a k-dimensional grid with $\mathtt{Pr}[u \to v] \propto d(u,v)^{-k}$, for any source–target pair $\langle s,t \rangle$ with $d(s,t) > cn$, the expected length of Greedy(s,t) is $\Omega(\log^2 n)$, where a dependence on the constant c is hidden by the $\Omega(\cdot)$. Because rank-based link probabilities differ by only a constant factor from $\mathtt{Pr}[s \to t] \propto d(s,t)^{-k}$ in uniform-population grids [16], we need only connect CautiousGreedy to Greedy to derive a lower bound.

Consider a uniform-population rank-based social network $\mathcal{R}^n_{uniform}$ where the underlying metric space is a ring (that is, we take $L = \{0, 1, \ldots, n\}$ and $d(i,j) = \min(|i-j|, |j-i|)$ and exactly one person living at each point). For a target person t, write $g_s(t) := \mathtt{E}[|\text{Greedy}(s,t)|]$ and $c_s(t) := \mathtt{E}[|\text{CautiousGreedy}(s,t)|]$ to denote the expected length of the paths found by the two algorithms to a particular target t. (Here the expectation is taken only over the random choices of the rank-based friendships.)

Lemma 9. *In $\mathcal{R}^n_{uniform}$, (i) if $n/4 \geq d(s',t) \geq d(s,t)$, then $g_{s'}(t) \geq g_s(t)$; and (ii) if $d(s,t) \leq n/4$, then $c_s(t) \geq g_s(t)$.* □

(The proofs, both by induction on $d(s,t)$, are omitted due to space constraints; the proof of claim (ii) relies on claim (i).)

Theorem 10. *In $\mathcal{R}^n_{uniform}$, for any source s and a target t chosen uniformly at random,* CautiousGreedy(s,t) *has expected length $\Omega(\log^2 n)$.*

Proof. For any constant $c \in (0, 1/4)$, there are $\Omega(n)$ people t with $cn < d(s,t) \leq n/4$. Thus with $\Omega(1)$ probability, the random target t satisfies $cn < d(s,t) \leq n/4$.

For any such target t, the expected length of CautiousGreedy(s,t) satisfies $c_s(t) \geq g_s(t) = \Omega(\log^2 n)$ by Lemma 9(ii) and the aforementioned theorem of Martel and Nguyen [20], trivially adapted to handle the differences between the distance-based and rank-based models, which can affect probabilities by a constant factor. For a constant fraction of the choices of t, then, we have an $\Omega(\log^2 n)$ bound on the expected length of CautiousGreedy(s,t). □

6 Future Directions

We have shown that, in rank-based networks, cautious-greedy routing performs well, in expectation for a randomly chosen target, as long as the underlying metric space has small aspect ratio, small doubling dimension, and large depth of field. In particular, we have been able to improve by a $O(\log \Delta)$ factor the results on Greedy [16]. But two natural questions remain:

- Is the expected length of the path found by Greedy to a randomly chosen target also short, say $O(\log^2 n)$ in a $\Theta(1)$-dimensional grid under Manhattan distance? (Or can CautiousGreedy be much better than Greedy?)
- Do either Greedy or CautiousGreedy achieve short paths—say of length $O(\log^{\Theta(1)}(n+\Delta)\cdot f(\rho,\alpha))$—for an *arbitrary* target (in expectation only over the random construction of the network)?

In fact, these questions appear to be intimately connected. In Sect. 1, we gave an example in which Greedy appears to hurt itself by taking a long-range link that brings it very close to a "distracting" point of high population, which attracts a large fraction of the rank-based links from people that Greedy subsequently encounters. But it is not too hard to see that Greedy "escapes" from the shadow of a distracting point in polylog(n) steps, because there is a reasonable probability of increasing one's distance from the distraction by a factor of 3/2 at any step. However, it is an open question as to whether some adversarial construction of a set of distracting points of various sizes might cause a particular target to be hard to reach efficiently. This question appears to be closely related to the question of whether the distracting points that Greedy may encounter in fact substantially slow down its performance. (For example, a natural analogue to Lemma 5 does not appear to hold for Greedy unless the ball B is expanded, as was done in previous analysis [16], which had the carryover effect of the extra logarithmic factor.)

It would also be interesting to better understand the role of the depth of field of the metric space. It is known, for example, that for doubling dimension $\alpha = \omega(\log \log n)$, there is no decentralized routing algorithm that achieves polylogarithmic routing time for all pairs of nodes [10]. The interaction between doubling dimension and depth of field is an interesting direction for further study—for example, is there a similar lower bound in networks with low doubling dimension but very small depth of field?

References

1. Abraham, I., Bartal, Y., Neiman, O.: Local embeddings of metric spaces. In: Symposium on Theory of Computing (2007)
2. Abraham, I., Gavoille, C., Goldberg, A., Malkhi, D.: Routing in networks with low doubling dimension. In: International Conference on Distributed Computing Systems (2006)
3. Adamic, L., Adar, E.: How to search a social network. Social Networks 27(3), 187–203 (2005)
4. Barrière, L., Fraigniaud, P., Kranakis, E., Krizanc, D.: Efficient routing in networks with long range contacts. In: International Symposium on Distributed Computing (2001)
5. Dodds, P., Muhamad, R., Watts, D.: An experimental study of search in global social networks. Science 301, 827–829 (2003)
6. Duchon, P., Hanusse, N., Lebhar, E., Schabanel, N.: Could any graph be turned into a small world? Theor. Comp. Sci. 355(1), 96–103 (2006)
7. Duchon, P., Hanusse, N., Lebhar, E., Schabanel, N.: Towards small world emergence. In: Symposium on Parallelism in Algorithms and Architectures (2006)
8. Fraigniaud, P.: Greedy routing in tree-decomposed graphs. In: European Symposium on Algorithms (2005)
9. Fraigniaud, P., Gavoille, C., Paul, C.: Eclecticism shrinks even small worlds. In: Principles of Distributed Computing (2004)
10. Fraigniaud, P., Lebhar, E., Lotker, Z.: A doubling dimension threshold $\Theta(\log \log n)$ for augmented graph navigability. In: European Symposium on Algorithms (2006)
11. Girdzijauskas, S., Datta, A., Aberer, K.: On small world graphs in non-uniformly distributed key spaces. In: International Conference on Data Engineering (2005)
12. Kleinberg, J.: Navigation in a small world. Nature 406, 845 (2000)
13. Kleinberg, J.: The small-world phenomenon: An algorithmic perspective. In: Symposium on Theory of Computing (2000)
14. Kleinberg, J.: Small-world phenomena and the dynamics of information. In: Advances in Neural Information Processing Systems (2001)
15. Kleinberg, J.: Complex networks and decentralized search algorithms. In: International Congress of Mathematicians (2006)
16. Kumar, R., Liben-Nowell, D., Tomkins, A.: Navigating low-dimensional and hierarchical population networks. In: European Symposium on Algorithms (2006)
17. Lebhar, E., Schabanel, N.: Close to optimal decentralized routing in long-range contact networks. In: International Colloquium on Automata, Languages and Programming (2004)
18. Liben-Nowell, D., Novak, J., Kumar, R., Raghavan, P., Tomkins, A.: Geographic routing in social networks. Proceedings of the National Academy of Sciences 102(33), 11623–11628 (2005)
19. Manku, G., Naor, M., Wieder, U.: Know thy neighbor's neighbor: the power of lookahead in randomized P2P networks. In: Symposium on Theory of Computing (2004)
20. Martel, C., Nguyen, V.: Analyzing Kleinberg's (and other) small-world models. In: Principles of Distributed Computing (2004)
21. Milgram, S.: The small world problem. Psych. Today 1, 61–67 (1967)

22. Motwani, R., Raghavan, P.: Randomized Algorithms. Cambridge U. Press, Cambridge (1995)
23. Şimşek, O., Jensen, D.: Decentralized search in networks using homophily and degree disparity. In: International Joint Conference on Artificial Intelligence (2005)
24. Slivkins, A.: Distance estimation and object location via rings of neighbors. In: Principles of Distributed Computing (2005)
25. Watts, D., Dodds, P., Newman, M.: Identity and search in social networks. Science 296, 1302–1305 (2002)

Locating Facilities on a Network to Minimize Their Average Service Radius*

Davide Bilò[2], Jörg Derungs[2], Luciano Gualà[3], Guido Proietti[1,4], and Peter Widmayer[2]

[1] Dipartimento di Informatica, Università di L'Aquila, 67010 L'Aquila, Italy
[2] Institut für Theoretische Informatik, ETH, 8092 Zürich, Switzerland
[3] Dipartimento di Matematica, Università di Tor Vergata, 00133 Roma, Italy
[4] Istituto di Analisi dei Sistemi ed Informatica, CNR, 00185 Roma, Italy
{guala,proietti}@di.univaq.it,
{dbilo,derungs,widmayer}@inf.ethz.ch

Abstract. Let $G = (V, E)$ denote an undirected weighted graph of n nodes and m edges, and let $U \subseteq V$. The *relative eccentricity* of a node $v \in U$ is the maximum distance in G between v and any other node of U, while the *radius* of U in G is the minimum relative eccentricity of all the nodes in U. Several facility location problems ask for partitioning the nodes of G so as to minimize some global optimization function of the radii of the subsets of the partition. Here, we focus on the problem of partitioning the nodes of G into exactly $p \geq 2$ non-empty subsets, so as to minimize the sum of the subset radii, called the *total radius* of the partition. This problem can be easily seen to be NP-hard when p is part of the input, but when p is fixed it can be solved in polynomial time by reducing it to a similar partitioning problem. In this paper, we first present an efficient $O(n^3)$ time algorithm for the notable case $p = 2$, which improves the $O(mn^2 + n^3 \log n)$ running time obtainable by applying the aforementioned reduction. Then, in an effort of characterizing meaningful polynomial-time solvable instances of the problem when p is part of the input, we show that (i) when G is a tree, then the problem can be solved in $O(n^3p^3)$ time, and (ii) when G has bounded treewidth h, then the problem can be solved in $O(n^{4h+4}p^3)$ time.

Keywords: Facility location problems, Graph partition, Graph radius, Graph treewidth, NP-hardness.

1 Introduction

Generally speaking, a *facility location* problem requires to identify a set of distinguished points in a given space as locations for installing a set of facilities, so as to (ideally) minimize the total effort needed from a certain set of customers

* Work partially supported by the Research Project GRID.IT, funded by the Italian Ministry of University and Research, by the European Union under COST 295 (DYNAMO), and by the Swiss BBW under grant no. C.05.0047. Part of this work has been developed while the first and the third authors were visiting ETH.

T. Tokuyama (Ed.): ISAAC 2007, LNCS 4835, pp. 587–598, 2007.

to use the service provided by the facilities. One can define several variants in this class of problems, depending on various parameters, like the opening cost of a facility as a function of the selected site, or the nature of the embedding space, and many others. For a comprehensive survey on facility location problems, we refer the reader to [5,8,10].

In a graph-theoretic setting, which is of interest for this paper, *facility location* is the following problem family: Given an undirected graph $G = (V, E)$ of n nodes and m edges with positive weights, and given a positive integer $p \leq n$, return a set of p distinct nodes for installing the facilities and an assignment of each node to one of the facilities, in such a way that some cost function defined on the shortest paths in G between any node and its respective facility is minimized. We study the cost function that considers for each facility the distance of the farthest assigned node, and that takes the sum of these distance over all facilities. It is important that the distance here is measured as the length of a shortest path in the entire graph G.

1.1 Related Work

Up to now, the research on graph location problems was generally founded on the basic assumption that each demand node has to be associated with its *nearest* (w.r.t. to the underlying graph shortest-path distance) facility. Within this framework, the unquestionably most extensively studied problems are the *p-median* and the *p-center*, for which we refer the reader to [8,10,6,7,11,14]. The aforementioned assumption conforms to the traditional *customer-centric* approach, in which the objective is to minimize the maximum effort that a customer has to make for *using* the service supplied by a facility. Many practical situations, however, are *facility-centric*, in the sense that the set-up and the operational cost of a facility might depend upon the maximum effort that a facility has to make *to serve* a customer. Problems of this sort arise in robotics applications, in multipoint delivery service systems, and in radio antenna network design, where the desired reach of an antenna (implied by the distance of its farthest assigned customer) limits the choice of suitable antenna models. In either case, the total cost depends on the *service radius* of each single installed facility (i.e., the distance in G of the farthest assigned node), and thus constraining a facility to serve all of its *neighborhood* (i.e., the set of nodes closer to it than to any other facility) might result in an overall drawback.

The stimulating applications defined by this different perspective motivated a series of papers aiming to solve the most immediate facility-centric location problem, namely that in which the *total* service radius (i.e., the sum of the facility service radii) has to be minimized. Initially, the attention of researchers has focused on the geometric setting. Here the problem can be formulated as that of covering a given set of n points in a d-dimensional Euclidean space by means of a set of at most p circles of radius r_i, whose center positions can be constrained in several different ways. The objective function is to minimize the sum of r_i^α over all these circles, where α is a constant that is typically 1 or larger, depending on the boundary conditions. For a summary of results in this general

framework we refer the reader to a recent paper by Alt *et al.* [1]. However, for the case $\alpha = 1$, which is close in spirit to our setting, the complexity of the problem is still unknown. Moreover, two PTAS's in Euclidean spaces of constant dimension are known, for both the case in which the number of circles is left unspecified (i.e., any $p \leq n$ is acceptable) but the center positions are restricted to a given set of possible locations [9], and in the (more general) case in which the number of circles is bounded and centering a circle at any given point has a variable non-negative cost [2].

Moving from Euclidean to discrete metric spaces, the problem becomes that of aggregating the given input elements into a set of p *clusters*, so as to minimize the sum of cluster radii. In this setting, an algorithm is known that computes, for any given $\epsilon > 0$, an approximate solution within a factor of $3.504 + \epsilon$ of the optimum, in $n^{O(1/\epsilon)}$ time [4].

Finally, moving from metric spaces to graphs, recently in [12] the authors studied the so-called *p-radius problem*, which asks for partitioning the nodes of a graph into exactly p non-empty subsets, so as to minimize the sum of radii of the corresponding induced subgraphs. Note the important distinction from the problem we study in this paper: While here, a path from a node to its facility is allowed to pass through arbitrary other nodes (and facilities), the p-radius problem requires that the path from a node to its facility visit only nodes assigned to the same facility (that is, paths must remain inside subgraphs). The authors showed that this problem is NP-hard, when p is part of the input, even if the input graph is metric, and provided for the special case $p = 2$ an algorithm requiring $O(mn^2 + n^3 \log n)$ time, and for the general case $p > 2$ an algorithm requiring $O(n^{2p}/p!)$ time.

1.2 Our Results

In this paper, we therefore aim at locating p facilities on the nodes of a graph, and then assigning each node to a facility, in such a way that the total service radius (recall that this is the sum of all service radii) is minimized. For this problem, that we call the *p-Location with Minimum Total Radius* (p-LMTR, from now on) *problem*, we first show an equivalence (based on a polynomial-time reduction) with the p-radius problem, which immediately implies the NP-hardness for our problem when p is part of the input. Afterwards, we present an efficient $O(n^3)$ time algorithm for the case $p = 2$, which can be used to return, thanks to the aforementioned reduction, a solution for the 2-radius problem, without any additional time charge. In this way, we improve the result presented in [12] for this problem. Then, we focus our attention on the especially notable case in which the input graph is a tree, and we show that the problem can be efficiently solved in $O(n^3p^3)$ time (which can be actually improved to $O(n^2p^3)$ time if the given tree is unweighted). Finally, we further extend in a non-trivial way the techniques developed for trees to graphs with a bounded treewidth h, for which we show that our problem can be solved in $O(n^{4h+4}p^3)$ time.

The paper is organized as follows: after stating the problem in Section 2, in Section 3 we show the equivalence with the p-radius problem and we present the

efficient algorithm for the case $p = 2$; in Section 4, we analyze the case in which G is a tree, while finally in Section 5 we study the problem when G has bounded treewidth.

2 Problem Statement

Let $G = (V, E)$ be an edge-weighted graph with weight function $w : E \to \mathbb{R}^+$. For any two given nodes a, b in G, we denote by $\delta_G(a, b)$ the *distance* (i.e., the length of a shortest path) in G between a and b. Given a subset $U \subseteq V$ and a node $v \in U$, we define the *relative* (i.e., w.r.t. U) *eccentricity* of v in G to be $\varepsilon_G(v, U) = \max_{u \in U} \delta_G(u, v)$, and we denote by $r_G(U)$ the *radius* of U in G, defined as $r_G(U) = \min_{v \in U} \varepsilon_G(v, U)$. The term radius will also refer to the associated path. We say that a node $v \in U$ is a *center* of U in G if $\varepsilon_G(v, U) = r_G(U)$. The radius of G can now be expressed as $r_G(V)$.

The p-LMTR problem is formally defined as follows:

Input: An undirected weighted graph $G = (V, E)$, $w : E \to \mathbb{R}^+$, and $p \in \mathbb{N}$.

Output: A pair $\langle \mathcal{L}, \mathcal{P} \rangle$, where $\mathcal{L} = \{c_1, \ldots, c_p\} \subseteq V$ is a set of p distinct location nodes (these are the nodes where the facilities will be installed), and $\mathcal{P} = \{V_1, \ldots, V_p\}$ is a partition of V (meaning that all nodes in V_i are associated with the location node $c_i, i = 1, \ldots, p$), such that the following cost function is minimum:

$$\phi(G, \mathcal{L}, \mathcal{P}) = \sum_{i=1}^{p} \max_{v \in V_i} \{\delta_G(c_i, v)\} = \sum_{i=1}^{p} \varepsilon_G(c_i, V_i). \tag{1}$$

Notice that, for any given partition, the selected cost function implies that the best possible choice for a location node of any subset of the partition is exactly a center of that subset in G. Because a center of a subset is easy to find, from an algorithmic point of view, our problem reduces to that of finding a p-partition (i.e., a partition of size p) $\mathcal{P} = \{V_1, \ldots, V_p\}$ of V such that the following cost function is minimized:

$$\varphi(G, \mathcal{P}) = \sum_{i=1}^{p} r_G(V_i). \tag{2}$$

Thus, in the rest of the paper, we will adopt this viewpoint, and our algorithms will deal with the minimization of (2).

3 Solving Efficiently the 2-LMTR Problem

A problem related to ours is the *p-radius problem* [12], which asks for a p-partition of V such that $\sum_{i=1}^{p} r_{G[V_i]}(V_i)$ is minimized, where $G[V_i]$ denotes the subgraph of G induced by V_i. It is worth noticing that $r_{G[V_i]}(V_i)$ may be different from $r_G(V_i)$. In the following, we show how to exploit this relationship between the two problems in order to provide efficient algorithms for the 2-LMTR and the 2-radius problem. We start by proving the following:

Lemma 1. *Let $p \in \mathbb{N}$ and let $G = (V, E)$ be an undirected weighted graph having n nodes, m edges, and positive edge weights. Then any solution of value $\widetilde{\varphi}$ for the p-LMTR problem can be transformed in $O(np\,(m + n \log n))$ time into a solution $V'_1, \ldots, V'_k$ for the p-radius problem such that $\sum_{i=1}^{p} r_{G[V'_i]}(V'_i) \leq \widetilde{\varphi}$.*

Proof. Let $\mathcal{P} = \{V_1, \ldots, V_p\}$ be the partition associated with the given solution of the p-LMTR problem, and for each i, let c_i be a center of V_i. We compute for each c_i a shortest paths tree $S_G(c_i)$ of G rooted at c_i. Now, let T_i denote the minimal subtree of $S_G(c_i)$ spanning V_i, and let $V(T_i)$ be the node set of T_i. In the following, we consider T_i as rooted at c_i. It is easy to see that the radius of T_i is equal to the radius of V_i. Then, we have that $\sum_{i=1}^{p} r_{T_i}(V(T_i)) = \widetilde{\varphi}$. Note that trees T_i may share some nodes, hence their node sets do not identify a partition of V. Our idea is to built p trees $T'_1, \ldots, T'_p$ from $T_1, \ldots, T_p$ such that $V(T'_1), \ldots, V(T'_p)$ is a partition of V and, for every i, the radius of T'_i is not greater than the one of T_i.

We say that a node v is *used k times* if it belongs to k trees. We proceed in steps, until no node is used more than once. At each step, we consider node used more than once, say v. Let T_i, T_j be two trees each containing v, and let ℓ_i (resp., ℓ_j) be the length of a longest directed path in T_i (resp., T_j) starting at v and leading away from c_i, i.e., going to a leaf of T_i. W.l.o.g. we can assume that $\ell_i \leq \ell_j$. Then we remove the subtree of T_i rooted at v from T_i, and we add the corresponding nodes to $V(T_j)$. Then, we compute the shortest paths tree $S_{G[V(T_j)]}(c_j)$, and the result is our new T_j. Notice that after having transformed T_i and T_j as described, the radii of both trees do not increase, and we have decreased the number of time this node is used by at least 1.

Concerning the time complexity, observe that the initial phase is dominated by the computation of all the centers, that can be accomplished in $O(mn + n^2 \log n)$ time by solving the all-pairs shortest paths problem in G. Afterwards, any node $v \in V$ is used at most p times, and so the number of steps for eliminating shared nodes is bounded by $O(np)$. Finally, each step can be easily performed in $O(m + n \log n)$ time. The claim follows. □

Notice that, since any feasible solution for the p-radius problem is also a feasible solution for the p-LMTR problem with at most the same cost, from Lemma 1 we know that the optimum solutions to both problems have the same objective value. In particular, it follows that the p-LMTR problem is NP-hard when p is part of the input, and polynomial time solvable for constant values of p. Moreover, it is not too hard to see that the $O(1)$-approximation algorithm in [4] can be used to obtain the same approximation ratio for the p-LMTR problem. Indeed, we can compute an approximate solution for the p-LMTR problem by running the algorithm in [4] on the metric closure of the input graph. Then, from Lemma 1, it follows that the same approximation ratio holds for the p-radius problem as well.

Now, we focus on the case $p = 2$, and we provide an efficient algorithm for both the p-LMTR problem and the p-radius problem. This improves the result in [12].

Theorem 1. *Both the* 2*-LMTR and the* 2*-radius problem can be solved in* $O(n^3)$ *time.*

Proof. Let us concentrate on the 2-LMTR problem. Our algorithm is the following. We start by solving the all-pairs shortest paths problem in G. Moreover, for each node v, we sort all the other $n-1$ nodes in a non-decreasing order of distance from v. Both operations can be accomplished in $O(mn+n^2 \log n)$ time.

Then, we consider all the possible pairs of nodes in G, since each of these pairs is a feasible pair for the subset centers. For each fixed pair, we compute an optimal bipartition of V w.r.t. the fixed pair, namely a distribution of the nodes into two subsets in such a way that the sum of the radii (computed w.r.t. the two fixed centers) of the two subsets is minimum. Finally, out of all feasible pairs, we select a pair minimizing the sum of the radii.

Given $c_1, c_2 \in V$, we can compute an optimal bipartition of V w.r.t. c_1, c_2 as follows. We consider $n-1$ (not necessarily distinct) values for the radius of the subset having center c_1, say $r_1, \ldots, r_{n-1}$, such that for each i the number of nodes having distance at most r_i from c_1 is at least i. In other words, let $c_1 = v_0, v_1, \ldots, v_{n-1}$ be the nodes ordered by non-decreasing distance from c_1, then we set $r_i = \delta_G(c_1, v_i)$. For each i from $n-1$ up to 0, we consider the partition $\{V_1^i = \{v_0, \ldots, v_i\} \setminus \{c_2\},\ V_2^i = V \setminus V_1^i\}$, and we compute its total radius. Since we have sorted the nodes by distances, we can find the sum of the radii of each partition in $O(1)$ time. Then finding an optimal bipartition w.r.t. a fixed pair c_1, c_2 takes $O(n)$ time, which implies that the overall time complexity of the algorithm for the 2-LMTR problem is bounded by $O(n^3)$. Moreover, notice that from Lemma 1, the same time complexity holds for the 2-radius problem as well, since the reduction takes $O(mn+n^2 \log n)$ time, and the claim follows. □

4 An Efficient Algorithm for Trees

In this section we provide an efficient algorithm for the p-LMTR problem for trees. To this aim, we focus on the p-radius problem on trees, and we provide a polynomial-time algorithm for solving it. Recall that the p-LMTR problem and the p-radius problem have the same objective value of an optimum solution, and hence the solution for the p-radius problem provided by the algorithm is an optimal solution for the p-LMTR problem as well.

Before describing the algorithm, we give the following definition. Let $G = (V, E)$ be a (weighted or unweighted) graph, and let $V_1, \ldots, V_p$ be a p-partition of V. We call each set of the p-partition a *cluster*. For a given subgraph G' of G, we say that a cluster V_i is *external to* G' (or simply *external*) if the center c_i of V_i does not belong to G', otherwise V_i is said to be *internal*.

The basic idea of the algorithm (whose pseudo-code is given below) is the following: given a weighted tree T and an integer $p \in \mathbb{N}$, we root T at any arbitrary node, and we consider each subtree of T in a bottom-up order. For each subtree T' with root x, we consider $O(np)$ subproblems, each of them described by a pair of values, say $\Delta \in \mathbb{R}$ and $k \in \mathbb{N}$, and denoted by $\Gamma_{T'}^{\Delta,k}$. Before giving the definition of $\Gamma_{T'}^{\Delta,k}$, we need some additional notations.

Algorithm 1 The p-LMTR algorithm for weighted trees.

Input: $T = (V, E), w : E \to \mathbb{R}^+, p \in \mathbb{N}$
Output: a p-partition of V minimizing the sum of the radii

```
root T at any arbitrary node x̄ ∈ V
for each x ∈ V in a bottom-up order do
   let T' be the subtree of T rooted at x
   for each k = 1, ..., p do
      find an optimal solution for Γ^{-1,k}_{T'}
   end for
   for each v ∈ V(T') \ {x} do
      Δ = δ_{T'}(x, v)
      for each k = 1, ..., p do
         find an optimal solution for Γ^{Δ,k}_{T'}
      end for
   end for
end for
return the solution for Γ^{-1,p}_{T}
```

Given a subtree T' with root x, and a value Δ, we define $V^{\Delta}_{T'} = \{v \in V(T') \mid \delta_{T'}(x, v) \leq \Delta\}$. Then, for a given $k \in \{1, \ldots, p\}$, we define $\Gamma^{\Delta,k}_{T'}$ to be the problem of finding a subset $U \subseteq V^{\Delta}_{T'}$ and a k-partition $\{V_1, \ldots, V_k\}$ of $V(T') \setminus U$, such that $\sum_{i=1}^{k} r_{G[V_i]}(V_i)$ is minimum. We refer to this problem as the problem of finding an optimal k-partition for T' *with option* Δ. The idea is that we aim at solving $\Gamma^{\Delta,k}_{T'}$ whenever we are considering a solution for T with $(p - k)$ clusters external to T', and in which each node $v \in V^{\Delta}_{T'}$ is within the radius of some cluster external to T'; this means that v can be picked either in some external cluster without increasing the sum of the radii, or in some cluster internal to T'. All $\Gamma^{\Delta,k}_{T'}$ with $\Delta < 0$ are equivalent, since all vertices of T' have to be picked in internal clusters. We use $\Gamma^{-1,k}_{T'}$ as a representative of this case.

Let T' be a subtree of T with root x. Now, we show how to solve $\Gamma^{\Delta,k}_{T'}$ for each $k = 1, \ldots, p$. If the number of nodes in $V(T') \setminus V^{\Delta}_{T'}$ is no more than k, then $\Gamma^{\Delta,k}_{T'}$ has a solution of cost 0, as we can make each node in $V(T') \setminus V^{\Delta}_{T'}$ a singleton cluster, and put all the remaining nodes in U. Otherwise, we have two cases:

Case $\Delta < 0$. Since x must belong to some cluster V_x internal to T', we consider all possible pairs of center and radius for V_x, and for each pair, say $\langle c, r\rangle$ (with $r \geq \delta_{T'}(c, x)$), we find an optimal k-partition for T' w.r.t. $\langle c, r\rangle$, and we select the one minimizing the sum of the radii. To compute the optimal k-partition for T' w.r.t. $\langle c, r\rangle$, we proceed as follows. We remove from T' the (unique) path in T' joining c and x, and we obtain a forest $\mathcal{F}$ consisting of a certain number of smaller subtrees.[1] Let $T_1, \ldots, T_h$ be these trees listed in any arbitrary order, and, for each $i = 1, \ldots, h$, let y_i be the root of T_i. Then,

[1] Notice that, if we put c and x in the same cluster V_x, then we have to put all the nodes on the path joining them in V_x, otherwise the radius of V_x would be unbounded.

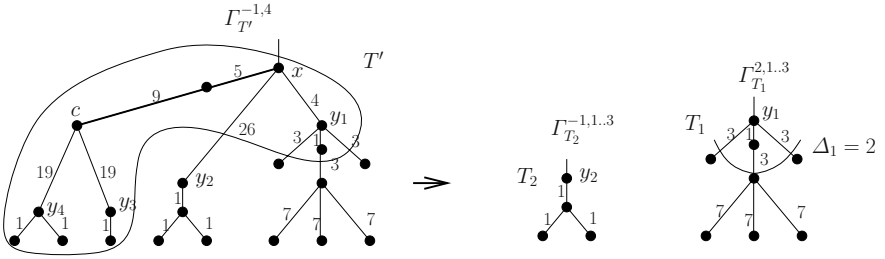


Fig. 1. An example of the execution of the algorithm for the problem $\Gamma_{T'}^{-1,4}$ for the case $\Delta < 0$ and x covered by an internal cluster of center c and radius 20

for each i, we consider the value $\Delta_i = r - \delta_{T'}(c, y_i)$. Observe that the nodes of T_i with distance at most Δ_i from y_i are the nodes that can be picked in V_x without increasing the radius of V_x, and therefore we only have to consider those trees with $V_{T_i}^{\Delta_i} \neq V(T_i)$, which cannot be placed completely within V_x. Moreover, notice that, for each i, we have already computed an optimal solution for $\Gamma_{T_i}^{\Delta_i,k'}$ for $k' = 1, \ldots, p$.[2] Hence, we can combine these solutions by means of a quite standard dynamic programming technique in order to obtain an optimal k'-partition for $\mathcal{F}$. Due to the lack of space, we do not present this dynamic programming algorithm here, and we refer the reader to the technical report. Notice that once we have computed an optimal k'-partition for $\mathcal{F}$, this in turn provides an optimal $(k'+1)$-partition for T' w.r.t. to $\langle c, r\rangle$. More formally, let k_i be the number of centers used for T_i in the optimal k'-partition for $\mathcal{F}$, and let U_i be the subset of nodes of T_i of the solution for $\Gamma_{T_i}^{\Delta_i,k_i}$. Then, the optimal $(k'+1)$-partition for T' w.r.t. $\langle c, r\rangle$ consists of all cluster of each k_i-partition for $\Gamma_{T_i}^{\Delta_i,k_i}$, plus the cluster V_x containing all the nodes in the path between x and c, and all nodes in $\bigcup_{i=1}^{h} U_i$. In Figure 1, we show an example of the execution of the algorithm.

Case $\Delta \geq 0$. We take the best solution between the two following ones: (i) the optimal solution picking x in some cluster internal to T', and (ii) the optimal solution picking x in some cluster external to T'. The computation of these solutions is described below.

(i) In this case, since x must belong to some internal cluster, then every node of T' must be picked in some internal cluster, otherwise there would exist some cluster external to T' with unbounded radius. Hence, $\Gamma_{T'}^{\Delta,k}$ is equivalent to $\Gamma_{T'}^{-1,k}$, that we have already solved.

(ii) Observe that, in this case, the problem $\Gamma_{T'}^{\Delta,k}$ is equivalent to $\Gamma_{T'}^{-1,k+1}$ w.r.t. $\langle c := x, r := \Delta\rangle$, for which we have already computed an optimal

[2] Indeed, we have already computed the optimal solution for $\Gamma_{T_i}^{\Delta_v,k'}$, for each k' and each $\Delta_v = \delta_{T_i}(y_i, v), v \in V(T_i)$, but it is not too hard to see that, if we denote by $v_1, v_2, \ldots$ the nodes of T_i sorted by non-decreasing distance from y_i, the optimal solution for $\Gamma_{T_i}^{\Delta_i,k'}$ coincides with the optimal solution for $\Gamma_{T_i}^{\Delta_{v_j},k'}$, where j is the maximum index such that $\Delta_{v_j} \leq \Delta_i < \Delta_{v_{j+1}}$.

solution. Hence, this solution can be available in constant time if, when we are considering the case $\Delta < 0$, we explicitly store the solution for $\Gamma_{T'}^{-1,k+1}$ w.r.t. $\langle c, r\rangle$, whenever $c = x$.

As far as the correctness of the algorithm is concerned, this can be shown by proving that, for each subtree T' of T with root x, the algorithm finds an optimal solution for $\Gamma_{T'}^{\Delta_v,k}$, $\Delta_v = \delta_{T'}(x, v), v \in V(T'), k = 1, \ldots, p$. This in turn can be proved by induction on the height of T', and immediately follows by construction.

Concerning the time complexity, the number of all possible pairs $\langle c, r\rangle$ for V_x in Step 5 is bounded by $O(n^2)$. If the obtained forest $\mathcal{F}$ has more than p trees, then there is no solution for $\Gamma_{T'}^{-1,p}$ w.r.t. $\langle c, r\rangle$. Otherwise, we can distribute the $k = 1, \ldots, p$ centers to $\mathcal{F}$ in $O(p^3)$, and thus Steps 4-6 can be accomplished in $O(n^2p^3)$ time, since by evaluating the center of the cluster containing x in depth-first order with increasing radius, it is possible to access the $\Gamma_{T_i}^{\Delta_i,k_i}$ at amortized constant time, even though the Δ_i are not discrete. Steps 7-12 take $O(np)$ time, since the number of different values of Δ is bounded by $n+1$ (including $\Delta < 0$), and since an optimal solution for $\Gamma_{T'}^{\Delta,k}$ for each k and each Δ can be found in $O(1)$ time. Then, since the number of managed subtrees of T is n, we have that the overall time complexity of the algorithm is bounded by $O(n^3p^3)$. Hence, because every optimal solution for the p-radius is also an optimal solution for the p-LMTR, we can state the following

Theorem 2. *For weighted trees, both the p-LMTR problem and the p-radius problem can be solved in $O(n^3p^3)$ time.* □

4.1 How to Reduce the Time Complexity for Unweighted Trees

On unweighted trees, we can use the following property to reduce the time complexity of the p-partition algorithm.

Lemma 2. *Let $G = (V, E)$ be an unweighted graph and let $p \in \mathbb{N}$. Then, there exists an optimal solution for the p-LMTR problem whose associated partition $\mathcal{P}$ satisfies the following property: $\forall i \neq j$ with $|V_i| \neq 1$ and $|V_j| \neq 1$, we have $\delta_G(c_i, c_j) > r_i + r_j$, where c_i, c_j and r_i, r_j are the centers and the radii of V_i and V_j, respectively.*

Proof. Let $\mathcal{P} = \{V_1, \ldots, V_p\}$ be a p-partition of an optimal solution not satisfying the property; then, we transform it into another optimal solution which satisfies the property by repeatedly using the following argument. Let V_i and V_j with $|V_i| \neq 1$ and $|V_j| \neq 1$, such that $\delta_G(c_i, c_j) \leq r_i + r_j$. Then there is a vertex c' on a path between c_i and c_j with $\delta_G(c_i, c') \leq r_j$ and $\delta(c', c_j) \leq r_i$. Every vertex $u \in V_i$ has distance at most $\delta_G(u, c_i)+\delta_G(c_i, c') \leq r_i+r_j$ from c'; correspondingly, $\delta_G(u, c') \leq r_j + r_i$ for all $u \in V_j$. Therefore we can replace V_i and V_j with V' and V'', where V'' is a single vertex from $V_j \setminus \{c'\}$ and $V' = V_i \cup V_j \setminus V''$. This replacement preserves both the number of sets and the sum of the radii. □

To find an optimal p-partition of an unweighted tree with the property specified in Lemma 2, we can ignore all $\Gamma_{T'}^{\Delta,k}$ with $\Delta \geq 0$, since every vertex u with $\delta_T(u, c_k) \leq r_k$ must be in V_k. Correspondingly, the cluster V_x with center c that contains the root x of T' has radius $r = \delta'_T(c, x)$, since any larger r would overlap with the radius of the cluster containing the parent of x. Thus, for unweighted trees, we can remove Steps 7-12 from Algorithm 1, and Steps 4-6 can be accomplished in $O(np^3)$ time. The overall solution is obtained by computing $\Gamma_T^{-1,p}$ with all possible centers c and radii $r \geq \delta_T(c, \bar{x})$ in $O(n^2p^3)$ time, thus resulting in an overall time complexity of $O(n^2p^3)$. Hence we can state the following

Theorem 3. *For unweighted trees, both the p-LMTR problem and the p-radius problem can be solved in $O(n^2p^3)$ time.* □

5 A Polynomial-Time Algorithm for Graphs with Bounded Treewidth

In this section we sketch a dynamic programming algorithm for the p-radius problem for graphs with bounded treewidth. The algorithm is based on the same techniques used in Section 4. Before explaining how the algorithm works, we recall the concepts of tree-decomposition and treewidth of a graph which have been introduced in [13].

Definition 1. *A* tree-decomposition *of a graph $G = (V, E)$ is a pair $\langle \{X_i \mid i \in I\}, T = (I, F)\rangle$ with $\{X_i \mid i \in I\}$ a family of subsets of V, one for each node of T, and T a tree such that*

- $\bigcup_{i \in I} X_i = V$;
- *for all edges $(u, v) \in E$, there exists an $i \in I$ with $u \in X_i$ and $v \in X_i$;*
- *for all $i, j, k \in I$: if j is on the path from i to k in T, then $X_i \cap X_k \subseteq X_j$.*

The *treewidth of a tree-decomposition* $\langle \{X_i \mid i \in I\}, T = (I, F)\rangle$ is $\max_{i \in I} |X_i| - 1$. The *treewidth of a graph* G is the minimum treewidth over all possible tree-decompositions of G. Given a subtree T' of the tree of a tree-decomposition of G, we will denote by $G_{T'}$ the subgraph of G induced by $\bigcup_{j \in V(T')} X_j$. For every edge $(i, j) \in F$, let T_1, T_2 be the two subtrees of T obtaining by removing the edge (i, j). By definition of tree-decomposition of G, observe that $Y = X_i \cap X_j$ separates the graph into 2 components G'_1, G'_2, where, G'_1 is the graph induced by $V(T_1) \setminus Y$, while G'_2 is the graph induced by $V(T_2) \setminus Y$, respectively.

The algorithm is an extension of the dynamic programming algorithm given in Section 4, with the main difference that it traverses over the nodes of the tree decomposition instead of the graph vertices. Given the tree T (rooted at any vertex) of a tree-decomposition $\langle \{X_i \mid i \in I\}, T = (I, F)\rangle$ of a graph G with treewidth h,[3] we consider each subgraph $G_{T'}$ induced by a subtree T' of T in a bottom-up order. For each subtree T' with root i, we consider $O((p + h)n^{2h+2})$

[3] Notice that T is part of the input. For graphs with constant treewidth, a tree-decomposition of G can be computed in linear time [3].

subproblems, each of them parametrized with the number of clusters and with a cluster per vertex in X_i. More precisely, let $[c, r]$ be the *cluster vector* associating with every vertex $x_j \in X_i$ a cluster with center c_j and radius r_j that covers x_j. Then, we define $\Gamma_{T'}^{[c,r],k}$ to be the subproblem of finding a k-partition $V_1, \ldots, V_k$ of $G_{T'}$ such that $\sum_{i=1}^{k} r_{G[V_i]}(V_i)$ is minimum, under the restriction that vertex $x_j \in X_i$ is in a cluster with center c_j and radius r_j.[4] Observe that, since every non-leaf X_i is a separating set and we are solving k-radius problem, we can simulate an external cluster $\langle c_j, r_j \rangle$ covering a vertex $x_j \in X_i$ with a cluster $\langle c'_j := x_j, r'_j := r_j - \delta_G(c_j, x_j) \rangle$ which still covers x_j. This implies that we do not need to take into account the *option* Δ we used for weighted trees. On the other hand, we need to compute $\Gamma_{T'}^{[c,r],k}$ for all $k \leq p + h$, since one external cluster may be simulated with up to $h + 1$ internal ones.

Assume that a vertex i of T is fixed, and assume that T' is the subtree of T with root i. For all the values $k = 1, \ldots, p + h$, the algorithm explores the space of all possible cluster vectors $[c, r]$ for X_i, and it finds an optimal solution for the subproblem $\Gamma_{T'}^{[c,r],k}$. So, assume that $[c, r]$ and k are fixed. Now, we show how the algorithm computes an optimal solution for $\Gamma_{T'}^{[c,r],k}$. For the sake of clarity, let us assume that node i has only one child, say ℓ, in T. We will deal with the case in which i has more than one child later. W.l.o.g, assume that some vertices of $G_{T'}$ are not covered by any cluster in the cluster vector $[c, r]$. Denote by T'' the subtree of T with root ℓ. As every vertex $x_j \in X_i$ is covered by the cluster with center c_j and radius r_j, then all the vertices in $Y = X_i \cap X_\ell$ are already covered. As observed above, since Y separates the vertices in $V(G_{T'}) \setminus Y$ from those in $V \setminus V(G_{T'})$, then we need to cover the vertices of $G_{T'}$ that we have not yet covered by using clusters which are internal to $G_{T''}$. Assume that the cluster vector $[c, r]$ contains exactly k' different centers. Hence, we take the best solution among the ones for the subproblems $\Gamma_{T''}^{[c',r'],k-k'}$, where each of these cluster vectors $[c', r']$ is such that, for every vertex in $X_i \cap X_\ell$, the corresponding cluster in $[c, r]$ and $[c', r']$ is identical.[5]. As a consequence, the time complexity for solving one subproblem in the case in which i has only one child, is $O(n^{2h})$.

The case in which i has $t > 1$ children, say $\ell_1, \ldots, \ell_t$, then, for every $j = 1, \ldots, t$ and for every $k' = 1, \ldots, p$, the algorithm selects the best solution among the ones for the subproblems $\Gamma_{T_j}^{[c',r'],k'}$, where (i) T_j is the subtree of T with root ℓ_j, and (ii) $[c', r']$ is a cluster vector such that, for every vertex in $X_i \cap X_{\ell_j}$, the corresponding cluster in $[c, r]$ and $[c', r']$ is identical. As observed at the beginning of this section, X_i separates the graph into t components $G'_1, \ldots, G'_t$, where, for each $j = 1, \ldots, t$, G'_j is the graph induced by $V(T_j) \setminus X_i$. Hence we can use again a standard dynamic programming technique (due to the lack of space, we refer the reader to the technical report) for combining the optimal solutions for the

[4] Clearly, if $\langle c_j, r_j \rangle = \langle c_t, r_t \rangle$ for some $j \neq t$, then we will consider it as one cluster, hence we will sum up $r = r_j = r_t$ to the objective function only once.

[5] Once again, if a cluster pair center-radius has more than one occurrence in all the cluster vectors we are considering, then it will contribute only once in the objective function of the problem $\Gamma_{T'}^{[c,r],k}$.

subproblems. In this case, the time complexity for solving all the subproblems for each child of i is $O(pn^{2h})$. Since a node i has at most $O(n)$ children [3], the overall time complexity for solving $\Gamma_{T'}^{[c,r],k}$ is $O(pn^{2h+1})$. Moreover, as we have to solve $O((p+h)n^{2h+2})$ subproblems per node in T, and since T has $O(n)$ nodes [3], we can state the following:

Theorem 4. *For undirected weighted graphs with treewidth h, both the p-LMTR problem and the p-radius problem can be solved in $O(n^{4h+4}p^2)$ time.* □

References

1. Alt, H., Arkin, E.M., Brönnimann, H., Erickson, J., Fekete, S.P., Knauer, C., Lenchner, J., Mitchell, J.S.B., Whittlesey, K.: Minimum-cost coverage of point sets by disks. In: Proc. 22nd ACM Symp. on Computational Geometry (SoCG 2006), pp. 449–458 (2006)
2. Biló, V., Caragiannis, I., Kaklamanis, C., Kanellopoulos, P.: Geometric clustering to minimize the sum of cluster sizes. In: Brodal, G.S., Leonardi, S. (eds.) ESA 2005. LNCS, vol. 3669, pp. 460–471. Springer, Heidelberg (2005)
3. Bodlaender, H.L.: A linear time algorithm for finding tree-decompositions of small treewidth. In: Proc. of the 25th ACM Symp. on Theory of Computing (STOC 1993), pp. 226–234 (1993)
4. Charikar, M., Panigrahy, R.: Clustering to minimize the sum of cluster diameters. J. of Computer and Systems Sciences 68(2), 417–441 (2004)
5. Drezner, Z., Hamacher, H.W.: Facility Location: Applications and Theory. Springer, Heidelberg (2004)
6. Kariv, O., Hakimi, S.L.: An algorithmic approach to network location problems, Part I: The p-centers. SIAM Journal on Applied Mathematics 37, 513–538 (1979)
7. Kariv, O., Hakimi, S.L.: An algorithmic approach to network location problems, Part II: The p-medians. SIAM Journal on Applied Mathematics 37(3), 539–560 (1979)
8. Labbé, M., Peeters, D., Thisse, J.F.: Location on networks. In: Ball, M., Magnanti, T., Francis, R.L. (eds.) Handbooks in Operations Research and Management Science: Network Routing, Elsevier, Amsterdam (1995)
9. Lev-Tov, N., Peleg, D.: Polynomial time approximation schemes for base station coverage with minimum total radii. Computer Networks 47(4), 489–501 (2005)
10. Mirchandani, P.B., Francis, R.L. (eds.): Discrete location theory. Wiley, New York (1990)
11. Megiddo, N., Tamir, A.: New results on the complexity of p-center problems. SIAM Journal on Computing 12(4), 751–758 (1983)
12. Proietti, G., Widmayer, P.: Partitioning the nodes of a graph to minimize the sum of subgraph radii. In: Asano, T. (ed.) ISAAC 2006. LNCS, vol. 4288, pp. 578–587. Springer, Heidelberg (2006)
13. Robertson, N., Seymour, P.D.: Graph minors. II. Algorithmic aspects of tree-width. J. Algorithms 7, 309–322 (1986)
14. Tamir, A.: An $O(pn^2)$ algorithm for the p-median and related problems on tree graphs. Operations Research Letters 19(2), 59–64 (1996)

Faster Combinatorial Algorithms for Determinant and Pfaffian*

Anna Urbańska

Institute of Computer Science
Warsaw University, Warsaw, Poland
ania@mimuw.edu.pl

Abstract. The algorithms of Mahajan and Vinay compute determinant and Pfaffian in a completely non-classical and combinatorial way, by reducing these problems to summation of paths in some acyclic graphs. The attractive feature of these algorithms is that they are division-free. We present a novel algebraic view of these algorithms: a relation to a pseudo-polynomial dynamic-programming algorithm for the knapsack problem. The main phase of MV-algorithm can be interpreted as a computation of an algebraic version of the knapsack problem, which is an alternative to the graph-theoretic approach used in the original algorithm. Our main results show how to implement Mahajan-Vinay algorithms without divisions, in time $\tilde{O}(n^{3.03})$ using the fast matrix multiplication algorithm.

Keywords: Algorithm, Determinant, Graph, Matrix, Pfaffian.

1 Introduction

Computation of a determinant is a very classical problem. The related concept is a Pfaffian of a matrix defined for skew-symmetric matrices. For such matrices the Pfaffian is the square root of the determinant, however it can be computed without involving the square root operation by replacing (in the definitions of the determinant) permutations by perfect matching. Pfaffians have important application in graph-matching problems, see [7].

The classical algorithm for computing the determinant is Gaussian elimination. It needs $O(n^3)$ (or $O(n^{2.38})$ if the fast matrix multiplication is used) additions, subtractions, multiplications and divisions. On the other hand, the explicit definition of the determinant as the sum of $n!$ products, shows that it can be computed *without* divisions. Avoiding divisions seems attractive when working over a commutative ring which is not a field, for example when the entries are integers, polynomials, or rational or even more complicated expressions. Such a computation of determinants arise in combinatorial problems, see [6]. Strassen [11] has given a general recipe for converting an algorithm that uses divisions to obtain an integral final result into an algorithm without divisions. Divisions are

* Supported by the grant of the Polish Ministery of Science and Higher Education N 206 004 32/0806.

T. Tokuyama (Ed.): ISAAC 2007, LNCS 4835, pp. 599–608, 2007.

simulated by expanding expressions into truncated power series. This is somehow reminiscent of the recursion which also involves power series that can be truncate because the result is a polynomial. Strassen's algorithm has the run time $O(n^5)$ (or $\tilde{O}(n^{3.38})$ if the fast matrix multiplication algorithm and the fast Fourier transformation algorithm are used).

In this paper we translate Mahajan and Vinay [9] beautiful graph-theoretic algorithm for division-free computation of determinant into a very simple $O(n^{3.5})$ algebraic algorithm and show how matrix multiplication is the key operation in our $\tilde{O}(n^{3.03})$ version. MV-algorithm originally work in $O(n^4)$ time and by Mahajan and Vinay was put to a different use, namely that of computing the characteristic polynomial [10] of a matrix without divisions, counting additions, subtractions and multiplications in the commutative ring of entries. Our result in the case of characteristic polynomial are in many ways analogous to those for the determinant. By the technique of Baur and Strassen [1] we obtain the adjoint of a matrix in the same division-free complexity.

We will also consider the Pfaffian of a skew-symmetric matrix, a quantity closely related to the determinant. Pfaffian arise naturally in the study of matchings, the Pfaffian of an oriented graph is just the sum over all possible perfect matchings except that each matching has an associated sign as well, dictated by the orientation. The results in the case of Pfaffian are in many ways analogous to those for the determinant.

We consider only to $n \times n$ integer matrices, but our algorithms apply to matrices over any commutative ring. The *determinant* of A, $det(A)$, is defined as

$$det(A) = (-1)^n \cdot \sum_{\sigma} sgn(\sigma) \cdot w(\sigma),$$

where the sum ranges over all permutations σ of the permutation group on $\{1, 2, \ldots, n\}$, S_n, $sgn(\sigma)$ is $(-1)^k$, where k is the number of cycles in cycle decomposition of σ and the *weight* of σ is $w(\sigma) = A[1, \sigma(1)] \cdot A[2, \sigma(2)] \cdot \ldots \cdot A[n, \sigma(n)]$.

The determinant of a skew-symmetric matrix ($A = -A^T$) is the square of another expression, which is called the *Pfaffian* [7]. Formally, the Pfaffian of a skew-symmetric matrix A with an even number n of rows and columns is defined as

$$Pf(A) = \sum_{\mathcal{M}} sgn(\mathcal{M}) \cdot w(\mathcal{M}),$$

where the sum ranges over all perfect matchings $\mathcal{M}$. Here, a *perfect matching* $\mathcal{M}$ is a partition of the set $\{1, 2, \ldots, n\}$ into $m = n/2$ unordered pairs and is written as

$$\mathcal{M} = \{\{i_1, j_1\}, \{i_2, j_2\}, \ldots, \{i_m, j_m\}\},$$

where, by convention, $i_k < j_k$ for each $k = 1, 2, \ldots, m$ and $i_1 < i_2 < \ldots < i_m$. The *sign* of the matching $\mathcal{M}$, $sgn(\mathcal{M})$, is define as the sign of permutation

$$\sigma_{\mathcal{M}} = \begin{pmatrix} 1 & 2 & 3 & 4 & \cdots & n-1 & n \\ i_1 & j_1 & i_2 & j_2 & \cdots & i_m & j_m \end{pmatrix}.$$

The *weight* of $\mathcal{M}$ is

$$w(\mathcal{M}) = A[i_1, j_1] \cdot A[i_2, j_2] \cdot \ldots \cdot A[i_m, j_m].$$

2 Algebraic View of MV Algorithm for Determinant

We introduce a *modified knapsack problem*: we have n types of items, for each type (i) we have m items of type (i). The item $Item_{i,j}$ has weight j and value $A[i, j]$. We have to choose a subset $\{Item_{i_1,j_1}, Item_{i_2,j_2}, \ldots, Item_{i_k,j_k}\}$ of the set of items, each type should be represented at most once. The total value should be maximum and total weight should equal n.

In other words we have to choose from each row of A at most one element, such that sum $j_1 + j_2 + \ldots + j_k$ of column indices equals n and the sum $A[i_1, j_1] + A[i_2, j_2] + \ldots + A[i_k, j_k]$ of chosen elements is maximal.

2.1 Algebraic Knapsack Problem

The modified knapsack problem can be formulated in more algebraic terms as follows. Assume we have two operations $\oplus$, $\otimes$ of a semi-ring S. For a rectangular $n \times m$ matrix A denote by $\mathcal{K}_t(A)$ the sum (with respect to $\oplus$) of all products $A[i_1, j_1] \otimes A[i_2, j_2] \otimes \ldots \otimes A[i_k, j_k]$, where $1 \le i_1 < i_2 < \ldots < i_k \le n$, $1 \le j_1, j_2, \ldots, j_k \le m$ and $j_1 + j_2 + \ldots + j_k = t$.

If $\oplus = \max$ and $\otimes = +$, then $\mathcal{K}_t(A)$ is the maximum value of taking a set of items from a set of n items, with constraints that the total sum of weights of these items is t and the values of the items are given by the table A. Hence this is the *knapsack problem*. If A is an integer $n \times n$ matrix and the operations $\oplus$, $\otimes$ are classical arithmetic operation $+$, $\cdot$ then we denote

$$Knapsack(A) = \mathcal{K}_n(A).$$

In our version of MV-algorithm we have two phases, the first corresponds to a variation of the transitive closure, called in the paper *SkewClosure*. The second one corresponds to an algebraic generalization of the knapsack problem.

2.2 Skew Closure of the Matrix

We can think of A as a weighted directed complete graph G_A in which the vertices are labeled $\{1, 2, \ldots, n\}$ and each edge (i, j) has weight $A[i, j]$. Let $\pi = (i_0, i_1, \ldots, i_k)$ be a path of length k in G_A. The *weight* of π is the product of the weights of the edges it contains

$$weight(\pi, A) \;=\; A[i_0, i_1] \cdot A[i_1, i_2] \cdot \ldots \cdot A[i_{k-1}, i_k].$$

We say that π is a *closed walk* with a *head* h if $i_0 = i_k = h$ and $i_t > h$ for each $0 < t < k$. By $\Pi_{h,k}$ we denote the set of all closed walks of length k with a head h and by $\hat{A}[h, k]$ the sum of the weights of all closed walks from $\Pi_{h,k}$, i.e.

$$\hat{A}[h, k] \;=\; \sum_{\pi \in \Pi_{h,k}} weight(\pi, A).$$

The matrix $\hat{A}$ is called the *skew-closure* of A and denoted by $SkewClosure(A)$.

Our version of MV algorithm, using the knapsack operation, can be formulated as follows.

Algorithm $ComputeDeterminant(A)$
/* Algebraic version of MV-algorithm */
$\hat{A} := SkewClosure(A)$;
return $(-1)^n \cdot Knapsack(-\hat{A})$;
end;

Mahajan and Vinay [9] have shown that $det(A)$ can be defined in terms of sum of paths in a certain (layered) directed acyclic graph. We can reformulate the theorem of Mahajan and Vinay in an equivalent algebraic formulation.

Theorem 1 (Correctness Theorem). *The algorithm* $ComputeDeterminant(A)$ *is correct:*

$$det(A) = (-1)^n \cdot Knapsack(-SkewClosure(A)).$$

Proof. A *closed walk sequence* is an ordered sequence of closed walks whose total length is n and whose heads are in strictly increasing order. The *weight* in A of a closed walk sequence $\mathcal{C}$ is the product of the weights of the edges it contains, i.e.

$$weight(\mathcal{C}, A) = \prod_{(i,j)\in\mathcal{C}} A[i,j].$$

The *sign* of $\mathcal{C}$, $sgn(\mathcal{C})$ is $(-1)^k$, where k is the number of closed walks in $\mathcal{C}$. Note that the sum of $(-1)^n \cdot sgn(\mathcal{C}) \cdot weight(\mathcal{C}, A)$ over only those closed walk sequences that are cycle covers of the graph is exactly the determinant of A. Moreover, Mahajan and Vinay in [9] show that in fact

$$det(A) = (-1)^n \cdot \sum_{\mathcal{C}} sgn(\mathcal{C}) \cdot weight(\mathcal{C}, A),$$

where the sum is over *all* closed walk sequences.

We group the closed walk sequences based on heads and on the lengths of the individual closed walks. In a closed walk sequence $\mathcal{C}$, if the length of closed walk C with head h is l, then any closed walk with head h and length l can replace C in $\mathcal{C}$ and still give a closed walk sequence. Because $\hat{A}[h,l]$ is the total weight of all closed walks which have a vertex h as a head and length l, then $(-1)^n \cdot det(A)$ is the sum of all products $(-1)^k \cdot \hat{A}[h_1,l_1] \cdot \hat{A}[h_2,l_2] \cdot \ldots \cdot \hat{A}[h_k,l_k]$, where $1 \le h_1 < h_2 < \ldots < h_k \le n$, $1 \le l_1, l_2, \ldots, l_k \le n$, $l_1 + l_2 + \ldots + l_k = n$ and this is exactly $Knapsack(-\hat{A}) = Knapsack(-SkewClosure(A))$. □

2.3 Computing *Knapsack*(*A*)

Let $A_{[k]}$ be the matrix A with only the first k rows taken. The classical computation $\mathcal{K}_t(A)$ is done by *dynamic programming* using the table $Table[i,j] = \mathcal{K}_j(A_{[i]})$. $Table(1,j) = A(1,j)$ for all j and for i,j from 1 to n we compute

$$Table[i,j] \;=\; \oplus_{0<p<j}\ (\ Table[i-1,p] \otimes A[i,j-p]\ )\ \oplus Table[i-1,j].$$

Consequently we have:

Lemma 1. *If A is an $n \times n$ matrix then $Knapsack(A)$ can be computed in $O(n^3)$ time using operations of summation and multiplication.*

3 Speeding-Up the Algorithm for Determinant

First we describe the simple version of our algorithm computing $SkewClosure(A)$ which works in $O(n^{3.5})$ time. We do not consider here the use of the fast matrix multiplication algorithm.

We have to compute the entries of the matrix $\hat{A} = SkewClosure(A)$, where $\hat{A}[h,k]$ is the sum of the weights of all closed walks of length k with a head h. Denote by A_h the matrix A with all rows and columns not greater than h zeroed, i.e. $A_h[i,j] = A[i,j]$ if $i > h$ and $j > h$, 0 otherwise and by a_h (a^h) the h^{th} row (column) of the matrix A with the first h entries zeroed. We look for the sequence of ring elements

$$\hat{A}[h,k] = a_h \cdot A_h^{k-2} \cdot a^h,$$

for $h = 1,2,\ldots,n$ and $k = 2,3,\ldots,n$ ($\hat{A}[h,1] = A[h,h]$).

We can compute this as follows. Let $\gamma = \lceil\sqrt{n}\rceil$, $\eta = \lceil n/\gamma\rceil$ and denote $u_h^{[q]} = a_h \cdot A_h^{q\cdot\gamma}$, $v_h^{[r]} = A_h^r \cdot a^h$, $Z_h = A_h^\gamma$, $w_h^{[r]} = a'_h \cdot A_{h-1}^r$, where a'_h is the h^{th} row of A with the first $h-1$ entries zeroed and let $v_h^{[-1]}[i] = w_h^{[-1]}[i] = 1$ if $i = h$, 0 otherwise. Z_n is the all-zeroes matrix.

Algorithm *SkewClosure*(A)
  for $h = n-1, n-2, \ldots, 1$ **do**
STEP 1 **for** $r = 0,1,\ldots,\gamma$ **do**
    $v_h^{[r]} \leftarrow A_h \cdot v_h^{[r-1]}$; $w_h^{[r]} \leftarrow w_h^{[r-1]} \cdot A_{h-1}$;
STEP 2 $Z_h \leftarrow Z_{h+1} + \sum_{k=0}^{\gamma}(v_{h+1}^{[k-1]} \cdot w_{h+1}^{[\gamma-k-1]})$;
STEP 3 **for** $q = 1,2,\ldots,\eta$ **do** $u_h^{[q]} \leftarrow u_h^{[q-1]} \cdot Z_h$;
STEP 4 **for** $r = 0,1,\ldots,\gamma$ **do**
    for $q = 0,1,\ldots,\eta$ **do** $\hat{A}[h, q\cdot\gamma + r + 2] \leftarrow u_h^{[q]} \cdot v_h^{[r]}$;
  return $\hat{A}$;
end;

We shall analyze our algorithm. The costly steps are Step 1, Step 2 and Step 3 and each of them works in time $O(n^{2.5})$. Step 4 need only $O(n^2)$ time and is dominated by this cost. Consequently our algorithm has the time complexity $O(n^{3.5})$.

The correctness proof is implicit in Fig. 1 and Fig. 2. [!h]

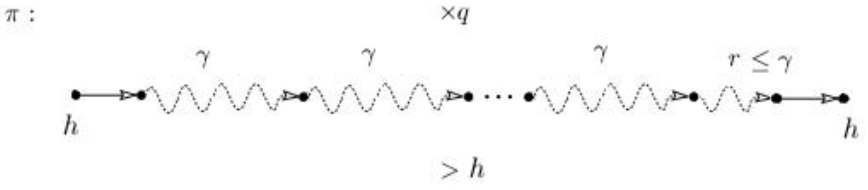


Fig. 1. Splitting of the path π of length k from a vertex h to h by vertices greater than h into two paths of length 1, q paths of length γ and a path of length r, where $k = q \cdot \gamma + r + 2$, $0 \leq r < \gamma$

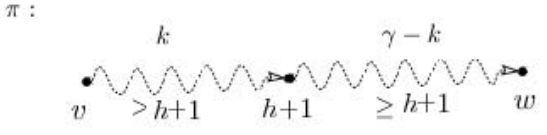


Fig. 2. Unique splitting of the path π of length γ from a vertex v to w by vertices greater than h, but including a vertex $h+1$ into a path of length k from v to $h+1$ by vertices *greater* than $h+1$ and a path of length $\gamma - k$ from $h+1$ to w by vertices *greater or equal* $h+1$. The paths by vertices greater than h (see Step 2 of the algorithm $SkewClosure(A)$) can be divided into two kind of paths, namely the paths by vertices greater than $h+1$ and that ones including a vertex $h+1$. The second one we divide with respect to the first position of $h+1$ on this path.

3.1 Using the Fast Matrix Multiplication Algorithm

As stated in the Introduction, sub-cubic matrix multiplication algorithms can be employed to improve the theoretical complexity of the above approach. Now let ω be the exponent for fast matrix multiplication algorithm. By Coppersmith and Winograd (1990) we may set $\omega = 2.3755$, see [3]. The considerations in this section are of a purely theoretical nature.

The γ here is no longer than n^ϵ, but n^ϵ, and ϵ is a constant which will be specified at the end.

Note that in Step 1 the vector $v_h^{[r]} = A_h \cdot v_h^{[r-1]}$ is the same as a vector $A \cdot v_h^{[r-1]}$ with the first h entries zeroed. Denote by $V^{[r]}$ the $n \times n$ matrix with columns $v_1^{[r]}, v_2^{[r]}, \ldots, v_n^{[r]}$ for $r = 0, 1, \ldots, \gamma$ and by $\nabla(M)$ the matrix M with the diagonal and all entries over it zeroed. Then we have

$$V^{[0]} = \nabla(A); \quad V^{[r+1]} = \nabla(A \cdot V^{[r]}).$$

Similarly the vector $w_h^{[r]} = w_h^{[r-1]} \cdot A_{h-1}$ is the same as a vector $w_h^{[r-1]} \cdot A$ with the first $h-1$ entries zeroed. Let now $W^{[r]}$ be the $n \times n$ matrix with rows $w_1^{[r]}, w_2^{[r]}, \ldots, w_n^{[r]}$ for $r = 0, 1, \ldots, \gamma$ and $\Delta(M)$ be the matrix M with all entries below diagonal zeroed. Then

$$W^{[0]} = \Delta(A); \quad W^{[r+1]} = \Delta(W^{[r]} \cdot A).$$

Cost of computing all matrices $V^{[r]}$ and $W^{[r]}$ for $r = 0, 1, \ldots, \gamma$ is $O(\gamma \cdot n^\omega)$.

The accumulated change of Z_h in Step 2 is

$$v_{h+1}^{[-1]} \cdot w_{h+1}^{[\gamma-1]} + v_{h+1}^{[0]} \cdot w_{h+1}^{[\gamma-2]} + v_{h+1}^{[1]} \cdot w_{h+1}^{[\gamma-3]} + \ldots + v_{h+1}^{[\gamma-1]} \cdot w_{h+1}^{[-1]} = V_{h+1} \cdot W_{h+1},$$

where V_{h+1} is an $n \times \gamma$ matrix with columns $v_{h+1}^{[-1]}, v_{h+1}^{[0]}, \ldots, v_{h+1}^{[\gamma-1]}$ and W_{h+1} is a $\gamma \times n$ matrix with rows $w_{h+1}^{[\gamma-1]}, w_{h+1}^{[\gamma-2]}, \ldots, w_{h+1}^{[-1]}$. The matrix $V_{h+1} \cdot W_{h+1}$ can be computed using the fast matrix multiplication algorithm but as we shall show we do not need computed all these matrices. We compute matrices Z_h only for $h = n, n-\eta, \ldots, n-(\gamma-1)\cdot\eta$. We can do this simply in time $O(\gamma \cdot n^\omega)$ since

$$Z_h = Z_{h+\eta} + V_{h+\eta} \cdot W_{h+\eta} + V_{h+\eta-1} \cdot W_{h+\eta-1} + \ldots + V_{h+1} \cdot W_{h+1}$$

$$= Z_{h+\eta} + [V_{h+\eta} | V_{h+\eta-1} | \ldots | V_{h+1}] \cdot \begin{bmatrix} W_{h+\eta} \\ W_{h+\eta-1} \\ \vdots \\ W_{h+1} \end{bmatrix} = Z_{h+\eta} + V_{(h)} \cdot W_{(h)},$$

where $V_{(h)}$ and $W_{(h)}$ are $n \times n$ matrices.

In Step 3 we can apply a similar technique as in Step 1. Namely for $i = 1, 2, \ldots \eta$

$$u_{h+i}^{[q]} = u_{h+i}^{[q-1]} \cdot Z_{h+i} = u_{h+i}^{[q-1]} \cdot Z_{h+\eta} + u_{h+i}^{[q-1]} \cdot [V_{h+\eta} | V_{h+\eta-1} | \ldots | V_{h+i+1}] \cdot \begin{bmatrix} W_{h+\eta} \\ W_{h+\eta-1} \\ \vdots \\ W_{h+i+1} \end{bmatrix}$$

and the second part of this sum is the same as a vector $u_{h+i}^{[q-1]} \cdot V_{(h)}$ with the last $i \cdot \gamma$ entries zeroed multiplied by the matrix $W_{(h)}$. Denote by $U_h^{[q]}$ for $h = n, n-\eta, \ldots, n-(\gamma-1)\cdot\eta$ and $q = 0, 1, \ldots, \eta$ the $\eta \times n$ matrix with rows $u_{h+\eta}^{[q]}, u_{h+\eta+1}^{[q]}, \ldots, u_{h+1}^{[q]}$ and let $\nabla_\star(M)$ be the $\eta \times n$ matrix, such that $\nabla_\star(M)[i,j] = M[i,j]$ if $j \le \gamma \cdot (i-1)$, 0 otherwise. Then

$$U_h^{[q]} = U_h^{[q-1]} \cdot Z_{h+\eta} + \nabla_\star(U_h^{[q-1]} \cdot V_{(h)}) \cdot W_{(h)}.$$

Cost of computing $U_h^{[r]}$ is proportional to the cost of multiplying an $\eta \times n$ matrix by an $n \times n$ matrix. We now appeal to fast methods for rectangular matrices, see [2], which show how to multiply an $n \times n$ matrix by an $n \times \nu$ matrix in $\tilde{O}(n^{\omega-\theta} \cdot \nu^\theta)$ arithmetic operations (by blocking the $n \times n$ matrix into $(t \times t)$-sized blocks and the

$n \times \nu$ matrix into $(t \times t^{\zeta})$-sized blocks such that $n/t = \nu/t^{\zeta}$ and that the individual block products only take $\tilde{O}(t^2)$ arithmetic steps each), where $\theta = (\omega - 2)/(1 - \zeta)$ with $\zeta = 0.2946289$. So we can compute all matrices $U_h^{[r]}$ for $h = n, n - \eta, \ldots, n - (\gamma - 1) \cdot \eta$ and $q = 0, 1, \ldots, \eta$ in $\tilde{O}(n^{1+\omega-\theta} \cdot \eta^{\theta})$ time.

The whole time complexity of Steps 4 is dominated by the complexities of other steps.

Consequently the overall running time of the algorithm $SkewClosure(A)$ is $\tilde{O}(\gamma \cdot n^{\omega} + n^{1+\omega-\theta} \cdot \eta^{\theta})$ and for $\gamma = \lceil n^{0.65} \rceil$, $\eta = \lceil n/\gamma \rceil$ it is $\tilde{O}(n^{3.03})$. We have proven:

Lemma 2. *The skew-closure can be computed in* $\tilde{O}(n^{3.03})$ *time.*

Due to Lemmas 1, 2 and Theorem 1 we have:

Theorem 2 (Complexity-Analysis Theorem). $det(A)$ *can be computed in* $\tilde{O}(n^{3.03})$ *time without division.*

4 Pfaffian

Pfaffian arise naturally in the study of matchings, the Pfaffian of an oriented graph is just the sum over all possible perfect matchings except that each matching has an associated sign as well, dictated by the orientation. This gives it a flavour similar to that of a determinant. In the absence of the sign, they would calculate the number of the perfect matchings in a graph, a problem that is well-known to be in $\#P$ [12]. Also, in the case of special graphs, it is known that the graph may be oriented in such a way that all the terms of the Pfaffian turn out to be positive. This obviously means there would be no cancellation and hence the Pfaffian would count the number of perfect matchings in the underlying graph. Such orientations of graphs are called Pfaffian orientation.

The Pfaffian can be computed using divisions by an elimination procedure that is similar to Gaussian elimination for symmetric matrices. Alternatively, the square root of the determinant gives the Pfaffian up to the correct sign. Both approaches may be undesirable if divisions or square roots should be avoided. Knuth [5] gives a short history of Pfaffians and argues that Pfaffians are in some way more fundamental than determinant to which they are closely related.

Generalizing the combinatorial characterization of determinant Mahajan, Subramanya and Vinay [8] shown that $Pf(A)$ can be defined in terms of sum of paths in a certain (layered) directed acyclic graph, as well.

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

matrix A

5	6
13	14

Shrink(A)

5	6	7	8
1	2	3	4
13	14	15	16
9	10	11	12

ExchRows(A)

Let A be a skew-symmetric matrix ($A = -A^T$). Denote by $Shrink(A)$ the $n/2 \times n/2$ matrix $\Xi(A)$ with all odd rows deleted and all columns greater than

$n/2$ deleted, i.e. $\Xi(A)[i,j] = A[2i,j]$ for $1 \le i,j \le n/2$. Denote by $ExchRows(A)$ the matrix resulting by exchanging odd-even neighboring rows of A. These operations are illustrated in the figure above for an example matrix.

Algorithm *ComputePfaffian*(A)
$A_\star \leftarrow ExchRows(A)$;
for $i,j = 1,2,\ldots,n$ **do**
$A_\star[i,j] := (-1)^{i+1}\ A_\star[i,j]$;
$\hat{A}_\star := SkewClosure(A_\star)$; $\Xi(\hat{A}_\star) := Shrink(\hat{A}_\star)$;
return $Knapsack(-\Xi(\hat{A}_\star))$;

Theorem 3. *The algorithm ComputePfaffian*(A) *is correct:*
$Pf(A) = Knapsack(-Shrink(SkewClosure(A_\star)))$.

Proof. Mahajan, Subramanya and Vinay in [8] show that $Pf(A) = \sum_{\mathcal{C}} sgn(\mathcal{C}) \cdot weight(\mathcal{C}, A_\star)$, where the sum is over all closed walk sequences whose total length is $n/2$ and heads are even. As in the case of determinant we group the closed walk sequences based on heads and on the lengths of the individual closed walks. Because $\hat{A}_\star[h,l]$ is the total weight in $A_\star$ of all closed walks which have a vertex h as a head and length l, then $Pf(A)$ is the sum of all products $(-1)^k \cdot \hat{A}_\star[h_1,l_1] \cdot \hat{A}_\star[h_2,l_2] \cdot \ldots \cdot \hat{A}_\star[h_k,l_k]$, where $h_1,h_2,\ldots,h_k$ are even, $1 \le h_1 < h_2 \ldots < h_k \le n$, $1 \le l_1,l_2,\ldots l_k \le n/2$, $l_1+l_2\ldots+l_k = n/2$ and this is exactly $Knapsack(-Shrink(\hat{A}_\star)) = Knapsack(-Shrink(SkewClosure(A_\star)))$. □

Due to Lemmas 1, 2 and Theorem 3 we have:

Theorem 4. *$Pf(A)$ can be computed in $\tilde{O}(n^{3.03})$ time without division.*

5 Characteristic Polynomial and Adjoint

The technique of Mahajan and Vinay can be used as easily to compute *all* coefficients of the characteristic polynomial of A, see [10]. Similarly as before we can speed-up original MV algorithms computing the characteristic polynomial. We omit in this version the proof of the following result.

Theorem 5. *The characteristic polynomial of the matrix can be computed in $\tilde{O}(n^{3.03})$ time without division. By the results of Baur and Strassen [1] we obtain the adjoint of a matrix in the same division-free complexity.*

References

1. Baur, W., Strassen, V.: The Complexity of Partial Derivatives. Theoretical Comput. Sci. 22, 317–330 (1983)
2. Coppersmith, D.: Rectangular Matrix Multiplication Revisited. J. Complex 13(1), 42–49 (1997)

3. Coppersmith, D., Winograd, S.: Matrix Multiplication via Arithmetic Progressions. In: Proceedings of the nineteenth Annual ACM Conference on Theory of Computing, pp. 1–6 (1987)
4. Karpiński, M., Rytter, W.: Fast Parallel Algorithms for Graph Matching Problems. Oxford University Press, Oxford (1998)
5. Knuth, D.: Overlapping Pfaffians. Electron. J. Comb. 3, No. 2, article R5, 13 pp. Printed version: J. Comb. 3(2), 147–159 (1996)
6. Krattenthaler, C.: Advanced Determinant Calculus. Séminaire Lotharingien de Combinatoire B42, 67 (1999)
7. Lovász, L., Plummer, M.: Matching Theory. Ann. Discr. Math. vol. 29. North-Holland Mathematics Studies, vol. 121, Amsterdam (1986)
8. Mahajan, M., Subramanya, P., Vinay, V.: A Combinatorial Algorithm for Pfaffians. In: Asano, T., Imai, H., Lee, D.T., Nakano, S.-i., Tokuyama, T. (eds.) COCOON 1999. LNCS, vol. 1627, pp. 134–143. Springer, Heidelberg (1999)
9. Mahajan, M., Vinay, V.: A Combinatorial Algorithm for the Determinant. In: Proceedings of the Eight Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 1997 (1997)
10. Mahajan, M., Vinay, V.: Determinant: Combinatorics, Algorithms and Complexity. Chicago Journal of Theoretical Computer Science 5 (1997)
11. Strassen, V.: Vermeidung von Divisionen. Journal of Reine U. Angew Math. 264, 182–202 (1973)
12. Valiant, L.: The complexity of computing the permanent. Theoretical Computer Science 8, 189–201 (1979)

A Polynomial-Time-Delay and Polynomial-Space Algorithm for Enumeration Problems in Multi-criteria Optimization

Yoshio Okamoto[1] and Takeaki Uno[2]

[1] Department of Information and Computer Sciences, Toyohashi University of Technlology, Hibarigaoka 1–1, Tempaku, Toyohashi, Aichi, 441-8580, Japan
okamotoy@ics.tut.ac.jp
[2] National Institute of Informatics, Hitotsubashi 2–1–2, Chiyoda-ku, Tokyo, 101-8430, Japan
uno@nii.jp

Abstract. We propose a polynomial-time-delay polynomial-space algorithm to enumerate all efficient extreme solutions of a multi-criteria minimum-cost spanning tree problem, while only the bi-criteria case was studied in the literature. The algorithm is based on the reverse search framework due to Avis & Fukuda. We also show that the same technique can be applied to the multi-criteria version of the minimum-cost basis problem in a (possibly degenerated) submodular system. As an ultimate generalization, we propose an algorithm to enumerate all efficient extreme solutions of a multi-criteria linear program. When the given linear program has no degeneracy, the algorithm runs in polynomial-time delay and polynomial space. To best of our knowledge, they are the first polynomial-time delay and polynomial-space algorithms for the problems.

1 Introduction

The multi-criteria optimization is a vast field in optimization theory, operations research, and decision science. In a multi-criteria optimization problem, we usually need to enumerate the solutions which have a certain specified property, for example, the Pareto optimality or the efficiency.[1] See Ehrgott [3] for detail.

There have been two main streams in algorithm design for the multi-criteria optimization: exact approach and approximate approach. In the exact approach, the enumeration has to be exact, namely, all the solutions have to be output (without any duplication). For example, in the multi-criteria linear programming many exact algorithms have been proposed which enumerate all efficient extreme solutions or enumerate all efficient faces (see Ehrgott [3] and references therein). For bi-criteria combinatorial optimization problems, Ulungu & Teghem [8] proposed the so-called two-phase method which first determines the

[1] The word "efficient" is used differently in multi-criteria optimization and in algorithm theory. In multi-criteria optimization (or economics) efficiency is just another name for Pareto optimality. We hope that the reader is never confused.

T. Tokuyama (Ed.): ISAAC 2007, LNCS 4835, pp. 609–620, 2007.

extreme efficient solutions then enumerate the rest of the efficient solutions. On the other hand, in the approximate approach the enumeration is partial. See Zitzler, Laumanns & Bleuler [11] for example. A bit different approximate approach was done by Papadimitriou & Yannakakis [7], which has a certain approximation guarantee. See a recent short survey by Zaroliagis [10].

This work concentrates on the exact approach and we will exploit techniques from enumeration algorithmics. Despite many algorithm have been reported for multi-criteria optimization from the exact approach viewpoint, few of them have a certain theoretical guarantee of complexity. Observe that enumeration of the Pareto-optimal extreme solutions of a single-criteria linear program is equivalent to enumeration of the vertices of a convex polyhedron, and a recent result by Khachiyan, Boros, Borys, Elbassioni & Gurvich [5] shows that this problem admits no polynomial total time algorithm unless P = NP. This looks one of the obstructions for a theoretical investigation. So, we concentrate on a simpler problem to reveal the difficulty for the development of an algorithmic theory of multi-criteria enumeration problems.

As a sample problem, we study the multi-criteria minimum-cost spanning tree problem: given a connected undirected graph and several edge-cost functions, we have to find all spanning trees which minimize some convex combinations of the cost functions. In the multi-criteria optimization terminology, the outputs are exactly the solutions for all possible weighted sum scalarizations, and they correspond to the extreme efficient solutions. The determination of the extreme efficient solutions is a first step for complete enumeration of the efficient solutions, for example in the two-phase method [8].

We will compare two main methods in enumeration algorithmics. One is the binary partition method, and the other is the reverse search method. In the binary partition method, we recursively divide the solution space until we get trivial instances. In the reverse search method proposed by Avis & Fukuda [1], we implicitly define a rooted tree on the solutions to be enumerated, and traverse it.

We try to apply the two enumeration methods above to the multi-criteria minimum-cost spanning tree problem. For the binary partition method, we prove that a subproblem arising from a natural binary partition approach is NP-complete. This implies that an approach by the binary partition method seems difficult. On the other hand, with the reverse search method we design an algorithm which runs with polynomial-time delay and in polynomial space. This is the first algorithm for this problem with such a complexity guarantee.

Our reverse-search algorithm can be extended to the multi-criteria version of the minimum-cost base problem in matroids and submodular systems. Furthermore, a similar algorithm turns out to work for the multi-criteria linear programming. Although there have been many algorithms proposed for the multi-criteria linear programming, none of them has a performance guarantee as running with polynomial-time delay and in polynomial space (see Ehrgott [3] and references therein). Indeed, these algorithms store all the outputs as a list in the working memory to get rid of duplication, which looks a bottleneck for the efficiency. We may accomplish the polynomial-time delay by a small modification (for example,

using a balanced binary search tree instead of a list). However, it appears difficult for these algorithms to achieve the polynomial space by a small modification; namely, the essential improvement for memory usage is by far hard. On the other hand, our reverse-search algorithm can achieve both of the goals. This exhibits the power of the reverse search, and we hope that this work initiates a more fruitful connection of the multi-criteria optimization with the algorithms community.

The paper is organized as follows. In the next section, we give an introduction to enumeration algorithmics terminology and a concise description of the multi-criteria minimum-cost spanning tree problem. Section 3 discusses some existing methods to enumerate the spanning trees and observe how natural extensions of these methods fail. This includes the NP-completeness result of a natural subproblem arising from a binary partition method. Then in Section 4, we consider how we can overcome this issue, and design an algorithm running in polynomial-time delay and polynomial space with the reverse search method. Section 5 discusses a possible generalization of our reverse search algorithm to the multi-criteria linear programming. The final section concludes the paper with some open questions.

2 Preliminaries

An *enumeration problem* asks to output all objects, called *solutions*, which satisfy a given condition. To measure the efficiency of enumeration algorithms, we have to take into account the size of output (i.e., the number of solutions) explicitly since it could be exponentially large in terms of the size of input. An enumeration algorithm runs in *polynomial-time delay* if for any output object the next output object can be obtained in polynomial time in the size of input; it runs in *polynomial-space* if the working space it uses is bounded by a polynomial of the size of input. Note that we only count the working space, excluding the space for outputs. Intuitively speaking, the working space is a read/write memory and the output space is a write-only disk.

A *convex combination* of k functions $c_1, c_2, \ldots, c_k$ is a function $\sum_{i=1}^{k} \lambda_i c_i$ for some non-negative real numbers $\lambda_1, \lambda_2, \ldots, \lambda_k$ summing up to one. We call the vector $(\lambda_1, \ldots, \lambda_k)^\top \in \mathbb{R}^k$ of coefficients the *barycentric coordinate* of the combination.

Given a connected undirected graph $G = (V, E)$, a *spanning tree* of G is an edge subset $T \subseteq E$ of size $|V| - 1$ which embraces no cycle. For a non-negative edge-cost function $c\colon E \to \mathbb{R}_+$, a *minimum-cost spanning tree* of G with respect to c is a spanning tree T of G which minimizes the total cost $c(T) = \sum_{e \in T} c(e)$. We study the following problem.

Problem: MC-MCST
Input: a connected undirected graph $G = (V, E)$ and k distinct non-negative edge-cost functions $c_1, \ldots, c_k\colon E \to \mathbb{R}_+$
Enumerate: the spanning trees of G each of which is minimum-cost with respect to some convex combination of $c_1, \ldots, c_k$.

We call a spanning tree of G *feasible* if it is minimum-cost with respect to some convex combination of $c_1, \ldots, c_k$ (i.e., if it is to be output in MC-MCST).

3 Failed Attempts for Generalization by Straightforward Approaches

In this section, we first describe two existing methods for enumeration of spanning trees in a given connected graph, and observe why the straightforward generalizations of them to MC-MCST do not give efficient algorithms.

3.1 Binary Partition Method

Let us first look at a simple binary partition approach to enumerate all spanning trees in a given connected undirected graph $G = (V, E)$. First of all, we choose an arbitrary edge $e_1 \in E$ and classify the spanning trees of G into two groups: those containing e_1 and those not containing e_1. Then, we choose another arbitrary edge $e_2 \in E \backslash \{e_1\}$, and divide the groups similarly. This will give a recursion tree, and we stop the recursive call when the obtained group is ensured to contain no spanning tree. In this way, we can reduce redundant computation. The problem to decide whether a group contains a spanning tree can be formulated as "for disjoint subsets $E_1, E_2 \subseteq E$, does there exist a spanning tree of G which contains the edges in E_1 but does not contain any edges in E_2?" This can be solved in linear time.

To solve MC-MCST in the same way, we have to solve the following problem.

Problem: BinaryPartition
Input: a connected undirected graph $G = (V, E)$, two disjoint subsets $E_1, E_2 \subseteq E$ and k distinct non-negative edge-cost functions $c_1, \ldots, c_k \colon E \to \mathbb{R}_+$
Question: Does there exist a spanning trees of G which contains the edges in E_1 but does not contain any edges in E_2 and is minimum-cost with respect to some convex combination of $c_1, \ldots, c_k$.

If the problem BinaryPartition can be solved in polynomial time, then we can use the same binary partition strategy as above to obtain an algorithm to solve MC-MCST in polynomial-time delay and polynomial space. However, the following theorem shows that it is quite unlikely for us to achieve this goal.

Theorem 1. *The problem BinaryPartition is NP-complete.*

Proof. We can easily see the membership of the problem in NP. We show NP-hardness. To this end, we reduce the satisfiability problem (SAT) to BinaryPartition. An instance of SAT is given as a set of boolean variables $x_1, \ldots, x_n$ and a set of clauses $C_1, \ldots, C_m$ each of which consists of (possibly several) literals. Each literal is either a variable or its negation.

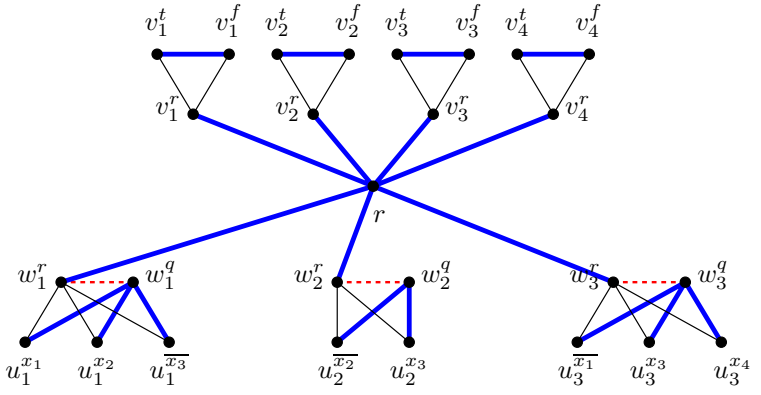


Fig. 1. Reduction in the proof of Theorem 1. This is an example for the formula $C_1 \wedge C_2 \wedge C_3$, where $C_1 = x_1 \vee x_2 \vee \overline{x_3}$, $C_2 = \overline{x_2} \vee x_3$. $C_3 = \overline{x_1} \vee x_3 \vee x_4$. A black thin edge belongs to $E \setminus (E_1 \cup E_2)$; a blue thick edge belongs to E_1; a red broken edge belongs to E_2.

Table 1. Summary of the costs

	$\{v_i^r, v_i^t\}$	$\{v_i^r, v_i^f\}$	$\{v_i^t, v_i^f\}$	$\{v_i^r, r\}$	$\{v_{i'}^r, v_{i'}^t\}$	$\{v_{i'}^r, v_{i'}^f\}$	$\{v_{i'}^t, v_{i'}^f\}$	$\{v_{i'}^r, r\}$
c_i	0	1	$1/n$	1	0	0	$1/n$	1
$c_{\overline{i}}$	1	0	$1/n$	1	0	0	$1/n$	1

	$\{w_j^r, w_j^q\}$	$\{w_j^r, r\}$	$\{w_j^q, u_j^\ell\}$	$\{w_j^r, u_j^\ell\}$
c_i	$2 - 1/(2n)$	1	1	1 if $\ell = x_i$, 2 otherwise
$c_{\overline{i}}$	$2 - 1/(2n)$	1	1	1 if $\ell = \overline{x_i}$, 2 otherwise

From the given instance of SAT, we construct a connected graph $G = (V, E)$. For each variable x_i we set up three vertices v_i^r, v_i^t, v_i^f. For each clause C_j we set up two vertices w_j^r, w_j^q, and for each literal ℓ of C_j we set up one vertex u_j^ℓ. We also use an extra vertex r. They are the vertices of G.

Next, we draw the edges of G. For each variable x_i, we draw edges $\{v_i^t, v_i^f\} \in E_1$, $\{v_i^r, v_i^t\} \in E \setminus (E_1 \cup E_2)$, $\{v_i^r, v_i^f\} \in E \setminus (E_1 \cup E_2)$, and $\{v_i^r, r\} \in E_1$. For each clause C_j we draw an edge $\{w_j^r, w_j^q\} \in E_2$, $\{w_j^r, r\} \in E_1$, and for each literal ℓ of C_j we draw edges $\{w_j^r, u_j^\ell\} \in E \setminus (E_1 \cup E_2)$, $\{w_j^q, u_j^\ell\} \in E_1$. This completes the description of G. Fig. 1 shows an example.

Now, we set up $2n$ cost functions, each of which is identified with a variable or its negation (i.e., a literal). Namely, for each positive literal x_i, we define the cost function c_i and, Similarly, for a negative literal $\overline{x_i}$, we define the cost function $c_{\overline{i}}$. The definition is as follows. They are summarized in Table 1: $c_i(\{v_i^r, v_i^t\}) = 0$, $c_i(\{v_i^r, v_i^f\}) = 1$, $c_i(\{v_i^t, v_i^f\}) = 1/n$, $c_i(\{v_i^r, r\}) = 1$; for every $i' \in \{1, \ldots, n\} \setminus \{i\}$, $c_i(\{v_{i'}^r, v_{i'}^t\}) = 0$, $c_i(\{v_{i'}^r, v_{i'}^f\}) = 0$, $c_i(\{v_{i'}^t, v_{i'}^f\}) = 1/n$, $c_i(\{v_{i'}^r, r\}) = 1$; for every $j \in \{1, \ldots, m\}$ and every literal ℓ of the clause C_j, $c_i(\{w_j^r, w_j^q\}) = 2 - 1/(2n)$, $c_i(\{w_j^r, r\}) = 1$, $c_i(\{w_j^q, u_j^\ell\}) = 1$, and

$$c_i(\{w_j^r, u_j^\ell\}) = \begin{cases} 1 & \text{if } \ell = x_i, \\ 2 & \text{otherwise.} \end{cases}$$

Similarly, $c_{\overline{i}}(\{v_i^r, v_i^t\}) = 1$, $c_{\overline{i}}(\{v_i^r, v_i^f\}) = 0$, $c_{\overline{i}}(\{v_i^t, v_i^f\}) = 1/n$, $c_{\overline{i}}(\{v_i^r, r\}) = 1$; for every $i' \in \{1, \ldots, n\} \setminus \{i\}$, $c_{\overline{i}}(\{v_{i'}^r, v_{i'}^t\}) = 0$, $c_{\overline{i}}(\{v_{i'}^r, v_{i'}^f\}) = 0$, $c_{\overline{i}}(\{v_{i'}^t, v_{i'}^f\}) = 1/n$, $c_{\overline{i}}(\{v_{i'}^r, r\}) = 1$; for every $j \in \{1, \ldots, m\}$ and every literal ℓ of the clause C_j, $c_{\overline{i}}(\{w_j^r, w_j^q\}) = 2 - 1/(2n)$, $c_{\overline{i}}(\{w_j^r, r\}) = 1$, $c_{\overline{i}}(\{w_j^q, u_j^\ell\}) = 1$, and

$$c_{\overline{i}}(\{w_j^r, u_j^\ell\}) = \begin{cases} 1 & \text{if } \ell = \overline{x_i}, \\ 2 & \text{otherwise.} \end{cases}$$

Thus, we complete the construction of an instance of MC-MCST.

We may prove that there exists a spanning tree of G which is minimum-cost with respect to some convex combination of the c_i and the $c_{\overline{i}}$, $i \in \{1, \ldots, n\}$ if and only if the given SAT instance is satisfiable. We omit the detail here due to the page limitation in this proceedings version. □

Hence, we give up adapting the binary partition method, and try another method.

3.2 Reverse Search Method

The reverse search method, proposed by Avis & Fukuda [1], is one of the most powerful techniques in enumeration algorithmics. Let $G = (V, E)$ be a given undirected connected graph, and we want to enumerate the spanning trees in G. To do this, we set up a rooted tree $\mathcal{R}$ on the spanning trees of G, namely, each node of $\mathcal{R}$ is a spanning tree of G. The enumeration will be done by traversing $\mathcal{R}$ in a depth-first-search manner, but we do not store the entire rooted tree itself; we just specify a parent-child relation which implicitly defines $\mathcal{R}$. In enumeration, we recursively move to children by the depth first search. Therefore, to design an efficient reverse-search algorithm it is enough for us to provide a parent-child relation so that we can find a parent/child efficiently. Since we do not need to store the entire family of spanning trees, but only a spanning tree under current investigation, this enables us to obtain an algorithm which runs in polynomial time delay and polynomial space. See Avis & Fukuda [1] and Nakano & Uno [6] for detail.

First of all, we define an adjacency relation on the family of spanning trees of G. Two distinct spanning trees T and T' of G are *adjacent* if the symmetric difference of T and T' is of size two. Through this adjacency relation, we naturally define the undirected graph $\mathcal{G}(G)$ which has the spanning trees of G as the node set. We can easily see that the number of nodes adjacent to one node is $O(|V||E|)$, and it is well-known [9, Exercise 2.1.62] that $\mathcal{G}(G)$ is connected.

On $\mathcal{G}(G)$ we define a rooted tree $\mathcal{R}$. For this purpose, we assume that the edges of G are labeled according to some fixed total order $\prec$ as $e_1 \prec e_2 \prec \cdots \prec e_m$.

Then, the root of $\mathcal{R}$ is defined as a (unique) lexicographically maximum spanning tree with respect to $\prec$, and a parent of a spanning tree T of G in the rooted tree $\mathcal{R}$ is a (unique) lexicographically maximum neighbor of T in $\mathcal{G}(G)$. This parent-child relation gives a well-defined rooted tree, and we can find a root, a parent of a non-root spanning tree, and the children of a non-leaf spanning tree in polynomial time. Therefore, this leads to an algorithm running in polynomial-time delay and polynomial space for enumerating the spanning trees in an undirected connected graph.

Let us try to generalize this approach to MC-MCST. We are given an undirected connected graph $G = (V, E)$ and k edge-cost functions $c_1, \ldots, c_k$. In this case, we consider the subgraph of $\mathcal{G}(G)$ induced by the feasible spanning trees (i.e., to be enumerated in MC-MCST). Denote this induced subgraph by $\mathcal{G}_M(G)$. Although $\mathcal{G}_M(G)$ depends on the edge-cost functions, we think them fixed thus do not include in the notation for convenience. Ehrgott [2] showed that the graph $\mathcal{G}_M(G)$ is always connected. Therefore, we can define a rooted tree $\mathcal{R}$ on $\mathcal{G}_M(G)$. The most natural way is to use the same strategy as in enumeration of the spanning trees of a connected graph. Namely, we assume that the edges of G are labeled according to some fixed total order $\prec$ as $e_1 \prec e_2 \prec \cdots \prec e_m$. Then, the root of $\mathcal{R}$ is defined as a (unique) lexicographically maximum spanning tree with respect to $\prec$, and a parent of a spanning tree T of G in the rooted tree $\mathcal{R}$ is a (unique) lexicographically maximum neighbor of T in $\mathcal{G}_M(G)$.

However, as opposed to the spanning trees enumeration case, for MC-MCST we have (at least) two troubles here. The first problem is that we do not know how to find a root in polynomial time. Actually, a greedy method fails since there can be many $\prec$-maximal feasible spanning trees in $\mathcal{G}_M(G)$. The second problem is even worse: the graph $\mathcal{R}$ may not be connected. Therefore, the rooted tree is not well-defined in general.

Hence, we need to devise another way to specify a rooted tree on $\mathcal{G}_M(G)$ if we wish to solve MC-MCST via reverse search.

4 The Proposed Algorithm

In our reverse-search algorithm for MC-MCST, we use $\mathcal{G}_M(G)$ defined in the previous section. Then, we have to define a promised rooted tree $\mathcal{R}$. For this purpose, we associate the following type of sequence to each feasible spanning tree. We assume that the edges of G are labeled according to some fixed total order $\prec$ as $e_1 \prec e_2 \prec \cdots \prec e_m$. This order $\prec$ will be used to break a possible tie. For a feasible spanning tree T of G, let $\boldsymbol{\lambda}_T \in \mathbb{R}^k$ be a lexicographically maximum barycentric coordinate of a convex combination of $c_1, \ldots, c_k$ which T minimizes. The following lemma shows that $\boldsymbol{\lambda}_T$ can be computed in polynomial time.

Lemma 2. *For every feasible spanning tree T of G, the vector $\boldsymbol{\lambda}_T$ can be found in polynomial time.*

Proof. We can phrase the problem in the following form.

$$\begin{aligned}
&\text{lex-max.} && \boldsymbol{\lambda} \\
&\text{subj. to} && \sum_{e\in T}\sum_{i=1}^{k}\lambda_i c_i(e) \le \sum_{e\in T'}\sum_{i=1}^{k}\lambda_i c_i(e) && \text{for all feasible } T' \text{ adjacent to } T, \\
& && \sum_{i=1}^{k}\lambda_i = 1, \\
& && \lambda_i \ge 0 && \text{for all } i \in \{1,\dots,k\}.
\end{aligned}$$

Note that in the first constraint we do not need to take into account all of the spanning trees of G, but we only need the spanning trees adjacent to T. This is due to the convexity (or matroid property) of the minimum-cost spanning tree problem (we omit the detail). Since the number of spanning trees adjacent to T is $\mathrm{O}(|V||E|)$, the number of constraints is polynomial. This lexicographic maximization problem can be solved by maximizing λ_i one by one in increasing order of $i \in \{1,\dots,k\}$, and each maximization is reduced to a linear program. Thus, using any polynomial-time algorithm for linear programming, we can solve the problem in polynomial time. □

The root of $\mathcal{R}$ is chosen as a feasible spanning tree R of G which has a lexicographically maximum $\boldsymbol{\lambda}_T$ among all feasible spanning tree T. Namely, such a barycentric coordinate $\boldsymbol{\lambda}_R$ should satisfy $(\lambda_R)_1 = 1$ and $(\lambda_R)_i = 0$ for all $i \in \{2,\dots,k\}$. Thus, R is a minimum-cost spanning tree with respect to c_1. If there are several minimum-cost spanning trees with respect to c_1, then we choose a $\prec$-maximum one as a root. Such a tree R is unique, and can be found in polynomial time by any polynomial-time minimum-cost spanning tree algorithm.

To specify the parent of a non-root feasible spanning tree T of G, we distinguish two cases. In the first case, we assume that $\boldsymbol{\lambda}_T = (1,0,0,\dots,0)^\top$. Then, T and R both minimize c_1. Therefore, as the following lemma certifies, we can obtain another minimum-spanning tree with respect to c_1 from T by deleting one edge from T and adding one edge from R.

Lemma 3. *Let $G = (V,E)$ be a connected undirected graph, $c\colon E \to \mathbb{R}_+$ be a non-negative edge-cost function, and $T_1, T_2 \subseteq E$ be minimum-cost spanning trees of G with respect to c. Then, there exist two edges $e_1 \in T_1 \setminus T_2$ and $e_2 \in T_2 \setminus T_1$ such that $(T_2 \cup \{e_1\}) \setminus \{e_2\}$ is also a minimum-cost spanning tree of G with respect to c.*

Although this is a well-known fact as, for example, in [9, Exercise 2.3.13], we give a proof here since we actually use the argument in the constructive proof below for the construction of our rooted tree.

Proof. Let us choose a minimum-cost edge $e_1 \in T_1 \setminus T_2$, namely $c(e_1) \le c(e)$ for every $e \in T_1 \setminus T_2$. Then, we can see that $T_2 \cup \{e_1\}$ embraces a unique cycle, say C. Note that C contains e_1. Now, we choose a maximum-cost edge

$e_2 \in C \setminus \{e_1\} \subseteq T \setminus \{e_1\}$, namely, $c(e_2) \geq c(e)$ for every edge $e \in C \setminus \{e_1\}$. Then, $T = (T_2 \cup \{e_1\}) \setminus \{e_2\}$ is a spanning tree of G.

Now we look at the cost. If $c(e_1) < c(e_2)$, then it follows that $c(T) = c(T_2) + c(e_1) - c(e_2) < c(T_2)$. Hence it contradicts the minimality of T_2. On the other hand, suppose that $c(e_1) > c(e_2)$. Then by the choice of e_1 it follows that $c(e) > c(e_2)$ for all $e \in T_1 \setminus T_2$. We consider a (unique) cycle C' in $T_1 \cup \{e_2\}$ and pick an arbitrary edge from $e' \in C' \setminus \{e_2\}$. Then, $T' = (T_1 \cup \{e_2\}) \setminus \{e'\}$ is a spanning tree of G and the cost is $c(T') = c(T_1) + c(e_2) - c(e') < c(T_1)$. Hence it contradicts the minimality of T_1. Thus, it must hold that $c(e_1) = c(e_2)$ and hence T is also a minimum-cost spanning tree of G with respect to c. □

The parent of T is constructively defined as follows. First we choose a minimum-cost edge $e_R \in R \setminus T$ (with respect to c_1), and if there are several choices, we choose a $\prec$-maximum one. This makes the choice of e_R unique. Then, $T \cup \{e_R\}$ contains a cycle C and we choose a maximum-cost edge $e_T \in C \setminus \{e_R\}$ (with respect to c_1), and if there are several choices, we choose a $\prec$-minimum one. From these choices, define the parent of T as $T' = (T \cup \{e_R\}) \setminus \{e_T\}$. From the discussion above, we can see that T' is a feasible spanning tree and $|R \triangle T'| < |R \triangle T|$. Note that T' can be found in polynomial time from T.

In the next case, we assume that $\boldsymbol{\lambda}_T \neq (1, 0, 0, \ldots, 0)^\top$. Then, we consider the corresponding $\boldsymbol{\lambda}_T$. Let $j \in \{2, \ldots, k\}$ be the minimum index such that $(\lambda_T)_j \neq 0$. Then, we take $\boldsymbol{\mu} \in \mathbb{R}^k$ obtained from $\boldsymbol{\lambda}_T$ by increasing the first component by a sufficiently small $\varepsilon > 0$ and decreasing the j-th component by ε. By our assumption for the second case, we can see that such an ε exists which keeps $\boldsymbol{\mu}_T$ to be a barycentric coordinate. Let S be a minimum-cost spanning tree of G with respect to $\sum_{i=1}^k \mu_i c_i$. If there are several minimum-cost spanning trees, then we choose a $\prec$-maximum one. By the lexicographic maximality of $\boldsymbol{\lambda}_T$ and the fact that $\boldsymbol{\mu}$ is lexicographically larger than $\boldsymbol{\lambda}_T$, we see that S is different from T. Since ε is sufficiently small, S is also a minimum-cost spanning tree with respect to $c = \sum_{i=1}^k (\lambda_T)_i c_i$. Hence, by Lemma 3 similarly to the first case, we choose an edge $e_S \in S \setminus T$ such that $c(e_S) \leq c(e)$ for all $e \in S \setminus T$ (if there are more than one such edges, then we choose the $\prec$-maximal one), and for a (unique) cycle C of $T \cup \{e_S\}$ we choose an edge $e_T \in C \setminus \{e_S\}$ such that $c(e_T) \leq c(e)$ for all $e \in C \setminus \{e_S\}$ (if there are more than one such edges, then we choose the $\prec$-minimal one). Then, we can see (from the proof of Lemma 3) that $T' = (T \cup \{e_S\}) \setminus \{e_T\}$ is a minimum-cost spanning tree with respect to c, and $|S \triangle T'| < |S \triangle T|$ holds. We define the parent of T as T', and we can find T' in polynomial time from T. In this way, the definition of a parent is completed. By the construction, the parent of T is adjacent to T in $\mathcal{G}_M(G)$, and it is unique. Furthermore, the next lemma is important.

Lemma 4. *Let $G = (V, E)$ be a connected undirected graph and $c_1, \ldots, c_k \colon E \to \mathbb{R}_+$ be non-negative edge-cost functions. Then, the parent-child relation defined above is well-defined. Namely, from a non-root feasible spanning tree $T \subseteq E$, by moving to the parent step by step we can arrive at the root R.*

Proof. Let T be a non-root feasible spanning tree and T' its parent. The investigation is divided into two parts according to the case distinctions above. Let us first consider when the first case is applied. In this case it holds that $\boldsymbol{\lambda}_{T'} = \boldsymbol{\lambda}_T = (1, 0, 0, \ldots, 0)^\top$ and $|R \triangle T'| < |R \triangle T|$. Therefore, we can arrive R at some point.

Next let us consider when the second case is applied. Let $T = T_0, T' = T_1$, and in general denote the parent of T_j by T_{j+1}. This construction can continue unless $\boldsymbol{\lambda}_{T_j} = (1, 0, 0, \ldots, 0)^\top$. Hence, it suffices to show that for every j there exists some $j' > j$ such that $\boldsymbol{\lambda}_{T_{j'}}$ is lexicographically larger than $\boldsymbol{\lambda}_{T_j}$. If this is true, then at some point (when the index is j, say) it must hold that $\boldsymbol{\lambda}_{T_j} = (1, 0, 0, \ldots, 0)^\top$ and the case is reduced to the first one.

Fix an arbitrary j. We are done if $\boldsymbol{\lambda}_{T_{j+1}}$ is lexicographically larger than $\boldsymbol{\lambda}_{T_j}$. Therefore, we assume $\boldsymbol{\lambda}_{T_{j+1}} = \boldsymbol{\lambda}_{T_j}$. Let S_j be a spanning tree used to obtain T_j as S was used to obtain T in the text above. Since S_j and S_{j+1} are dependent only on $\boldsymbol{\lambda}_{T_j}$ and $\boldsymbol{\lambda}_{T_{j+1}}$ respectively, it holds that $S_j = S_{j+1}$. However, for any i it holds that $|S_i \triangle T_{i+1}| < |S_i \triangle T_i|$. Therefore, there cannot be an infinitely long sequence $S_i = S_{i+1} = S_{i+2} = \cdots$ of identical spanning trees. Thus, there must exist some $j' > j$ such that $\boldsymbol{\lambda}_{T_{j'}}$ is lexicographically larger than $\boldsymbol{\lambda}_{T_j}$. □

From the discussion above, we finally obtain the following theorem.

Theorem 5. *By the reverse search algorithm described above, we can solve MC-MCST in polynomial-time delay and polynomial space.*

5 Generalization

The reverse search algorithm in the previous section can be generalized in several ways. A close inspection of the discussion shows that we only used the matroid property of the minimum-cost spanning tree problem in the algorithm. Therefore, we can conclude that the multi-criteria minimum-cost base problem in matroids can be solved in polynomial-time delay and polynomial space, when a matroid is given as the independent set oracle. More generally, we can solve the multi-criteria minimum-cost base problem in submodular systems in polynomial-time delay and polynomial space when a submodular function is given as a value-giving oracle. To this end, we need to identify the adjacent bases of a given base in a submodular system. This task is an instance of the submodular function minimization problem, which can be solved in polynomial time [4].

As an extreme generalization, we can consider the multi-criteria linear programming. In a *linear program*, we are given a system of inequalities $A\boldsymbol{x} \geq \boldsymbol{b}, \boldsymbol{x} \geq \boldsymbol{0}$ where $A \in \mathbb{R}^{m \times n}$ is a matrix, and $\boldsymbol{b} \in \mathbb{R}^m$ is a vector. Then we want to find, for a given $\boldsymbol{c} \in \mathbb{R}^n$, a solution $\boldsymbol{x}$ to the inequality system which minimizes $\boldsymbol{c}^\top \boldsymbol{x}$.

The inequality system above defines a convex polyhedron, called the *feasible region* of the problem. Here we assume (without loss of generality) that it is bounded and non-empty. With this assumption, a feasible region has at least one extreme point, and furthermore there exists an optimal solution which is

an extreme point of the polyhedron. We call such a solution an *extreme* optimal solution. In a *multi-criteria linear program*, we are given a system of linear inequalities $A\boldsymbol{x} \geq \boldsymbol{b}, \boldsymbol{x} \geq \boldsymbol{0}$, and we want to enumerate the extreme optimal solutions which minimize some convex combination of given k cost vectors $\boldsymbol{c}^1, \ldots, \boldsymbol{c}^k \in \mathbb{R}^n$.

Problem: MC-LP
Input: a matrix $A \in \mathbb{R}^{m \times n}$, two vectors $\boldsymbol{b} \in \mathbb{R}^m$ and $\boldsymbol{c}^1, \ldots, \boldsymbol{c}^k \in \mathbb{R}^n$
Enumerate: the extreme solutions $\boldsymbol{x}$ to the inequality system $A\boldsymbol{x} \geq \boldsymbol{b}, \boldsymbol{x} \geq \boldsymbol{0}$ which minimize some convex combination of $\boldsymbol{c}^1, \ldots, \boldsymbol{c}^k$.

We call an instance of MC-LP *non-degenerated* if every extreme point of the polyhedron determined by the given inequality system lies on n facets.

Theorem 6. *The non-degenerated MC-LP can be solved in polynomial-time delay and polynomial space.*

Proof (sketch). In the feasible region every extreme solution is adjacent to other extreme solutions through edges. This adjacency naturally defines an undirected graph, and in the same way as we did for MC-MCST we can implicitly specify a rooted tree in this graph. For a non-degenerated linear program, every extreme solution is adjacent to at most n other extreme solutions, and the adjacent extreme solutions can be found by pivot operations in polynomial time. The connectedness of the analogue of $\mathcal{G}_M(G)$ is known [3]. Furthermore, we can obtain propositions similar to Lemmas 2, 3 and 4 (the proofs are similar), and thus Theorem 6 is proven. □

Note that MC-LP with possible degeneracy seems very difficult to tackle. It is known that the vertex enumeration of a degenerated convex polyhedron, which corresponds to the enumeration of the extreme solutions to a single-criterion linear program, cannot be performed in polynomial total time (hence not in polynomial-time delay and polynomial space) unless P = NP [5].

6 Concluding Remark

We have looked at some multi-criteria optimization problems from the viewpoint of enumerative algorithmics. There seem many problems in multi-criteria optimization to which the algorithm theory can potentially contribute.

A key fact in our reverse search algorithm for MC-MCST is that there are at most polynomially many spanning trees adjacent to one spanning tree. This is no longer the case if we consider the bipartite matching problem. So far, we do not know how to obtain a polynomial-time delay and polynomial-space algorithm for the multi-criteria assignment problem (i.e., maximum bipartite matching problem). We can show that a natural binary partition approach does not work in the same way as we did in Section 3. We leave this issue as an open problem.

Another problem is concerned with Lemma 2, where we saw that $\boldsymbol{\lambda}_T$ can be obtained in polynomial time. However, it uses a polynomial-time linear programming algorithm, hence not a strongly polynomial-time algorithm. We do not know whether it can be computed in strongly polynomial time.

Acknowledgments. This work started when the authors visited Institute of Theoretical Computer Science, ETH Zurich in 2005. The authors are grateful to Emo Welzl and his group members for hospitality. A part of this research is supported by Grant-in-Aid for Scientific Research from Ministry of Education, Science and Culture, Japan, and Japan Society for the Promotion of Science.

References

1. Avis, D., Fukuda, K.: Reverse search for enumeration. Discrete Applied Mathematics 65, 21–46 (1996)
2. Ehrgott, M.: On matroids with multiple objectives. Optimization 38, 73–84 (1996)
3. Ehrgott, M.: Multicriteria Optimization, 2nd edn. Springer, Heidelberg (2005)
4. Grötschel, M., Lovász, L., Schrijver, A.: Geometric Algorithms and Combinatorial Optimization, 2nd edn. Springer, Berlin New York (1993)
5. Khachiyan, L., Boros, E., Borys, K., Elbassioni, K., Gurvich, V.: Generating all vertices of a polyhedron is hard. In: Proc. 17th SODA. Full version to appear in Discrete & Computational Geometry, pp. 758–765 (2006)
6. Nakano, S.-I., Uno, T.: Constant time generation of trees with specified diameter. In: Hromkovič, J., Nagl, M., Westfechtel, B. (eds.) WG 2004. LNCS, vol. 3353, pp. 33–45. Springer, Heidelberg (2004)
7. Papadimitriou, C., Yannakakis, M.: On the approximability of trade-offs and optimal access of web sources. In: Proc. 41st FOCS, pp. 86–92 (2000)
8. Ulungu, E.L., Teghem, J.: The two phase method: an efficient procedure to solve bi-objective combinatorial optimization problems. Foundations of Computing and Decision Sciences 20, 149–165 (1995)
9. West, D.B.: Introduction to Graph Theory, 2nd edn. Prentice Hall, Upper Saddle River (2001)
10. Zaroliagis, C.: Recent advances in multiobjective optimization. In: Lupanov, O.B., Kasim-Zade, O.M., Chaskin, A.V., Steinhöfel, K. (eds.) SAGA 2005. LNCS, vol. 3777, pp. 45–47. Springer, Heidelberg (2005)
11. Zitzler, E., Laumanns, M., Bleuler, S.: A tutorial on evolutionary multiobjective optimization. In: Gandibleux, X., Sevaux, M., Sörensen, K., T'kindt, V. (eds.) Metaheuristics for Multiobjective Optimisation, Lecture Notes in Economics and Mathematical Systems, vol. 535, pp. 3–38 (2004)

The Parameterized Complexity of the Unique Coverage Problem*

Hannes Moser[1,**], Venkatesh Raman[2], and Somnath Sikdar[2]

[1] Institut für Informatik, Friedrich-Schiller-Universität Jena,
Ernst-Abbe-Platz 2, D-07743 Jena, Germany
moser@minet.uni-jena.de
[2] The Institute of Mathematical Sciences,
C.I.T Campus, Taramani, Chennai 600113, India
{vraman,somnath}@imsc.res.in

Abstract. We consider the parameterized complexity of the Unique Coverage problem: given a family of sets and a parameter k, we ask whether there exists a subfamily that covers at least k elements exactly once. This *NP*-complete problem has applications in wireless networks and radio broadcasting and is also a natural generalization of the well-known Max Cut problem. We show that this problem is fixed-parameter tractable with respect to the parameter k. That is, for every fixed k, there exists a polynomial-time algorithm for it. One way to prove a problem fixed-parameter tractable is to show that it is kernelizable. To this end, we show that if no two sets in the input family intersect in more than c elements there exists a problem kernel of size k^{c+1}. This yields a k^k kernel for the Unique Coverage problem, proving fixed-parameter tractability. Subsequently, we show a 4^k kernel for this problem. However a more general weighted version, with costs associated with each set and profits with each element, turns out to be much harder. The question here is whether there exists a subfamily with total cost at most a prespecified budget B such that the total profit of uniquely covered elements is at least k, where B and k are part of the input. In the most general setting, assuming real costs and profits, the problem is not fixed-parameter tractable unless $P = NP$. Assuming integer costs and profits we show the problem to be $W[1]$-hard with respect to B as parameter (that is, it is unlikely to be fixed-parameter tractable). However, under some reasonable restriction, the problem becomes fixed-parameter tractable with respect to both B and k as parameters.

1 Introduction

In this paper, we consider the parameterized complexity of the Unique Coverage problem. This problem was introduced by Demaine et al. [2] as a natural maximization version of Set Cover and has applications in several areas including wireless networks and radio broadcasting. Unique Coverage is

* Supported by a DAAD-DST exchange program, D/05/57666.
** Supported by the Deutsche Forschungsgemeinschaft, project ITKO (iterative compression for solving hard network problems), NI 369/5.

T. Tokuyama (Ed.): ISAAC 2007, LNCS 4835, pp. 621–631, 2007.

defined as follows. Given a ground set $\mathcal{U} = \{1, 2, \ldots, n\}$, a family of subsets $\mathcal{S} = \{S_1, \ldots, S_t\}$ of $\mathcal{U}$ and a positive integer k, we ask whether there exists a subcollection $\mathcal{S}' \subseteq \mathcal{S}$ such that at least k elements are covered uniquely by the members in $\mathcal{S}'$. An element is covered uniquely if it appears in exactly one set of $\mathcal{S}'$. The optimization version requires to maximize the number of uniquely covered elements.

The weighted version of Unique Coverage is called Budgeted Unique Coverage and is defined as follows. Given a ground set $\mathcal{U} = \{1, 2, \ldots, n\}$, a profit p_i for each element $i \in \mathcal{U}$, a family of subsets $\mathcal{S}$ of $\mathcal{U}$, a cost c_i for each set $S_i \in \mathcal{S}$, a budget B and a positive integer k, we ask whether there exists a subset $\mathcal{S}' \subseteq \mathcal{S}$ such that the total cost of $\mathcal{S}'$ is at most B and such that the profit of the uniquely covered elements is at least k. The optimization version asks for a subset $\mathcal{S}'$ of total cost at most B such that the profit of uniquely covered elements is maximized.

The original motivation for this problem is a real-world application arising in wireless networks [2]. Assume that we are given a map of the densities of mobile clients along with a set of possible base stations, each with a specified building cost and a specified coverage region. The goal is to choose a set of base stations, subject to a budget on the total building cost, in order to maximize the density of served clients. The difficult aspect of this problem is the interference between base stations. A mobile client's reception is better when it is within the range of a few base stations. An ideal situation is when every mobile client is within the range of exactly one base station. This is the situation modelled by the Budgeted Unique Coverage problem. The Unique Coverage problem is closely related to a single "round" of the Radio Broadcast problem [1]. For more about this relation, see Demaine et al.'s work [2].

One can also view the Unique Coverage problem as a generalization of the Max Cut problem [2]. The input to the Max Cut problem consists of a graph $G = (V, E)$ and the goal is to find a cut (T, T'), where $\emptyset \neq T \subset V$ and $T' = V \setminus T$, that maximizes the number of edges with one endpoint in T and the other endpoint in T'. Let $\mathcal{U}$ denote the set of edges of G and for each vertex $v \in V$ define $S_v = \{e \in E : e \text{ is incident to } v\}$. Finally let $\mathcal{S} = \cup_{v \in V}\{S_v\}$. Then G has a cut (T, T') with at least k edges across it if and only if $\mathcal{S}' = \cup_{v \in T}\{S_v\}$ uniquely covers at least k elements of the ground set.

Known Results. (Budgeted) Unique Coverage was introduced by Demaine et al. [2]. They have considered the approximability of this problem. On the positive side, they give an $\Omega(1/\log n)$-approximation for Budgeted Unique Coverage. Moreover, if the ratio between the maximum cost of a set and the minimum profit of an element is bounded by B, then there exists an $\Omega(1/\log B)$-approximation. Concerning approximation hardness, they show that Unique Coverage is hard to approximate to within a factor of $O(1/\log^c n)$ for some constant $0 < c \leq 1$, and they strengthen this inapproximability to $O(1/\log n)$ based on a hardness hypothesis for Balanced Bipartite Independent Set.

Our Results. In this paper, we give first-time results on the parameterized complexity of Unique Coverage and Budgeted Unique Coverage. Compared

Unique Coverage (*Parameter: k*)	*Result*	Sect.
Each element occurs in at most b sets	$(k-1)b$ kernel	3.1
Intersection size bounded by c	k^{c+1} kernel	3.2
General	4^k kernel	3.3
Each set of size at most b	2^{b+k} kernel	3.3
Budgeted Unique Coverage		
Arbitrary costs/profits (pars. B and k)	Not FPT (unless P = NP)	4.1
Integer weights (par. B)	$W[1]$-hard	4.1
Integer weights (intersection number $f(k)$; pars. B and k)	$O^*((B \cdot 2^{f(k)})^{B+k})$-time algo.	4.2
Integer weights (pars. B and k)	Open	

Fig. 1. Main results in this paper

to the related Set Cover problem, which is W[2]-complete with respect to the number of sets in the solution as parameter[1], Unique Coverage becomes fixed-parameter tractable with respect to the number of uniquely covered elements. In other words, the number of uniquely covered elements seems to be a good parameter in order to exploit and reveal the inherent structure of coverage problems in general. Our results indicate that the budgeted variant, Budgeted Unique Coverage, is a much harder problem. More specifically, we show the following.

We show that a special case of Unique Coverage where any two sets in the input family intersect in at most c elements is fixed-parameter tractable by demonstrating a polynomial kernel of size k^{c+1}. This leads to a problem kernel of size k^k in the general case, proving that Unique Coverage is fixed-parameter tractable. However, the general case can be improved using results from extremal combinatorics on strong systems of distinct representatives to obtain a 4^k kernel. For the Budgeted Unique Coverage problem there are several variants. If the profits and costs are allowed to be arbitrary positive real numbers, then Budgeted Unique Coverage, with parameters B and k, is not fixed-parameter tractable unless $P = NP$. If we restrict the costs and profits to be positive integers and parameterize by B, then the problem is $W[1]$-hard. In the case when the number of sets intersecting any given set of the input family is bounded by a function of k, the problem is fixed-parameter tractable with parameters B and k. The main results of this paper along with the relevant sections in which they appear are depicted in Figure 1.

2 Preliminaries

We briefly introduce the necessary concepts concerning parameterized complexity. A parameterized problem is a subset of $\Sigma^* \times \mathbb{N}$, where Σ is a finite alphabet

[1] This can be shown by a reduction from Dominating Set [3,6].

and $\mathbb{N}$ is the set of natural numbers. An instance of a parameterized problem is therefore a pair (I, k), where k is the parameter. In the framework of parameterized complexity, the running time of an algorithm is viewed as a function of two quantities: the size of the problem instance *and* the parameter. A parameterized problem is said to be *fixed parameter tractable* (FPT) if there exists an algorithm for the problem with running time $f(k) \cdot |I|^{O(1)}$, where f is a computable function only depending on k.

A common method to prove that a problem is fixed-parameter tractable is to provide data reduction rules that lead to a problem kernel. A *data reduction rule* is a polynomial-time algorithm which takes a problem instance (I, k) and either

- outputs YES or NO according as (I, k) is a YES or a NO-instance, or
- replaces (I, k) by an equivalent instance (I', k') such that $|I'| \leq |I|$ and $k' \leq k$,

where two problem instances (I, k) and (I', k') are *equivalent* if they are both YES-instances or both NO-instances. An instance to which none of a given set of data reduction rules applies is called *reduced* with respect to this set of rules. A parameterized problem is said to have a *problem kernel* if the resulting reduced instance has size $f(k)$ for a function f depending only on k. If a parameterized problem has a kernel, then it is clearly fixed-parameter tractable. Simply use brute-force on the kernel to decide whether the given instance is a YES-instance or not.

A parameterized problem π_1 is *fixed-parameter reducible* to a parameterized problem π_2 if there exist functions $f, g : \mathbb{N} \to \mathbb{N}$, $\Phi : \Sigma^* \times \mathbb{N} \to \Sigma^*$ and a polynomial $p(\cdot)$ such that for any instance (I, k) of π_1, $(\Phi(I, k), g(k))$ is an instance of π_2 computable in time $f(k) \cdot p(|I|)$ and $(I, k) \in \pi_1$ if and only if $(\Phi(I, k), g(k)) \in \pi_2$. The basic complexity class for fixed-parameter intractability is $W[1]$ as there is strong evidence that $W[1]$-hard problems are not fixed-parameter tractable [3]. To show that a problem is $W[1]$-hard, one needs to exhibit a fixed-parameter reduction from a known $W[1]$-hard problem to the problem at hand. For more on parameterized complexity see [3,5].

We write $O^*(f(k))$ to denote a running time of $O(f(k) \cdot \text{poly}(n, k))$, where n is the input size and k is the parameter. That is, we use the $O^*(\cdot)$ notation to suppress polynomial factors in the running time.

3 The Unique Coverage Problem

Let $(\mathcal{U} = \{1, 2, \ldots, n\}, \mathcal{S} = \{S_1, S_2, \ldots, S_m\}, k)$ be an instance of UNIQUE COVERAGE. Apply the following data reduction rules on $(\mathcal{U}, \mathcal{S}, k)$ until no longer applicable.

R1. If there exists $S_i \in \mathcal{S}$ such that $|S_i| \geq k$, then the given instance is a YES-instance.

R2. If there exists $S_1, S_2 \in \mathcal{S}$ such that $S_1 = S_2$, then delete S_1.

These reduction rules are obviously correct. In the following we always assume that the given instance of UNIQUE COVERAGE is reduced with respect to the above rules.

3.1 Bounded Number of Occurrences

We begin with the simple case where each element $e \in \mathcal{U}$ is contained in at most b sets of $\mathcal{S}$. A special case of this situation is MAX CUT ($b = 2$).

Lemma 1. *If each element $e \in \mathcal{U}$ occurs in at most b sets of $\mathcal{S}$ then the* UNIQUE COVERAGE *problem admits a kernel of size $b(k-1)$.*

Proof. Find a maximal collection $\mathcal{S}'$ of pairwise disjoint sets in $\mathcal{S}$. If $|\cup_{S\in\mathcal{S}'} S| \geq k$, we are done. Therefore assume $|\cup_{S\in\mathcal{S}'} S| \leq k-1$. Since every set in $\mathcal{S} - \mathcal{S}'$ intersects some set in $\mathcal{S}'$ and since every element occurs in at most b sets in $\mathcal{S}$, we have $|\mathcal{S} - \mathcal{S}'| \leq (k-1)(b-1)$. But $|\mathcal{S}'| \leq k-1$ and so $|\mathcal{S}| \leq b(k-1)$. □

The proof of Lemma 1 applies a proof principle which is a basis for the proof of the following more complicated case.

3.2 Bounded Intersection Size

Consider the situation where for all $S_i, S_j \in \mathcal{S}$ we have $|S_i \cap S_j| \leq c$, for some constant c. In this case we say that the problem instance has *bounded intersection size c* and show that the problem admits a polynomial kernel of size $O(k^{c+1})$. First consider the case when $|S_i \cap S_j| \leq 1$.

Lemma 2. *Suppose that for all $S_i, S_j \in \mathcal{S}$, $i \neq j$, we have $|S_i \cap S_j| \leq 1$. If an element $e \in \mathcal{U}$ is covered by at least $k+1$ sets, then one can obtain a solution covering k elements uniquely in polynomial time.*

Proof. Suppose an element $e \in \mathcal{U}$ is covered by the sets $S_1, \ldots, S_{k+1}$. Then by reduction rule $R2$, at most one of these sets can have size 1. The remaining k sets uniquely cover at least one element each. □

One can now easily obtain a kernel of size k^2 for the case when the intersection size is at most 1.

Lemma 3. *Suppose that for all $S_i, S_j \in \mathcal{S}$, $|S_i \cap S_j| \leq 1$. If $|\mathcal{S}| \geq k^2$, then there exists $\mathcal{T} \subseteq \mathcal{S}$ that covers at least k elements uniquely.*

Proof. Greedily find a maximal collection $\mathcal{S}' = \{S_1, \ldots, S_p\}$ of pairwise disjoint sets in $\mathcal{S}$. Note that if $|\cup_{S_i\in\mathcal{S}'} S_i| \geq k$, then we are done. Therefore assume, $|\cup_{S\in\mathcal{S}'} S| \leq k-1$ (this also implies $p \leq k-1$). Since $|\mathcal{S}| \geq k^2$, and since every set in $\mathcal{S}$ intersects with at least one set in $\mathcal{S}'$, by the pigeonhole principle there exists an element $e \in \cup_{S\in\mathcal{S}'} S$ such that at least $k+1$ sets $T_1, \ldots, T_{k+1}$ in $\mathcal{S} - \{S_1, \ldots, S_p\}$ contain e. For otherwise, each element in $\cup_{S\in\mathcal{S}'} S$ is contained in at most k sets of $\mathcal{S} \setminus \mathcal{S}'$, which implies that $|\mathcal{S}| \leq (k-1)k + p < k^2$, a contradiction. By Lemma 2, this collection $\mathcal{T} = \{T_1, \ldots, T_{k+1}\}$ of $k+1$ sets uniquely covers at least k elements. □

Next, we generalize these observations to the case when $|S_i \cap S_j| \leq c$, for some constant c.

Theorem 1. *Suppose that for all $S_i, S_j \in \mathcal{S}$ we have $|S_i \cap S_j| \leq c$, for some positive constant c. If $|\mathcal{S}| \geq k^{c+1}$ then there exists $\mathcal{T} \subseteq \mathcal{S}$ that covers k elements uniquely.*

Proof. By induction on c. For $c = 1$, this follows from Lemma 3. Assume the theorem to hold for $c > 1$. Greedily obtain a maximal collection $\mathcal{S}' = \{S_1, \ldots, S_p\}$ of pairwise disjoint sets. If $|\cup_{S_i \in \mathcal{S}'} S_i| \geq k$ then we are done. Therefore assume $|\cup_{S \in \mathcal{S}'} S| \leq k-1$ (this also implies $p \leq k-1$). Since $|\mathcal{S}| \geq k^{c+1}$, and since every set in $\mathcal{S}$ intersects with at least one set in $\mathcal{S}'$, there exists $e \in \cup_{S \in \mathcal{S}'} S$ such that at least $k^c + 1$ sets in $\mathcal{S} - \{S_1, \ldots, S_p\}$ contain e. For otherwise, $|\mathcal{S}| \leq (k-1)k^c + p < k^{c+1}$, a contradiction. Let $T_1, \ldots, T_{k^c+1}$ be some $k^c + 1$ such sets. Delete e from each of these sets. We obtain at least k^c nonempty distinct sets $T'_1, \ldots, T'_{k^c}$ (there is at most one set consisting of the element e only which is deleted in this process). Note that any two of these sets intersect in at most $c-1$ elements. By induction hypothesis, there exists a collection $\mathcal{T}' \subseteq \{T'_1, \ldots, T'_{k^c}\}$ that uniquely covers at least k elements, and thus there exists a collection $\mathcal{T} \subseteq \{T_1, \ldots, T_{k^c}\}$ that uniquely covers at least k elements (just take the solution for T' and add e to every set in it). This proves the theorem. □

Corollary 1. Unique Coverage *admits a kernel of size k^{c+1} for bounded intersection size c.*

Note that $c \leq k-1$ and therefore for the general case we have a kernel of size k^k.

Corollary 2. *The* Unique Coverage *problem is fixed-parameter tractable and admits a problem kernel of size k^k.*

An algorithm that checks all possible subsets of a family of size k^k to see whether any of them uniquely covers at least k elements is an FPT algorithm with time complexity $O^*(2^{(k^k)})$. But note that we can assume without loss of generality that every set in the solution covers at least one element uniquely. Thus it suffices to check whether subfamilies of size at most k uniquely cover at least k elements. This can be done in time $O^*(k^{k^2}) = O^*(2^{k^2 \log k})$. However, this kernelization result is tailored especially for the bounded intersection size case. It turns out that a much better kernel can be obtained for the general case.

3.3 General Case

We now show that Unique Coverage has a kernel of size 4^k using a result on strong systems of distinct representatives. Given a family of sets $\mathcal{S} = \{S_1, \ldots, S_m\}$, *a system of distinct representatives* for $\mathcal{S}$ is an m-tuple $(x_1, \ldots, x_m)$ where the elements x_i are distinct and $x_i \in S_i$ for all $i = 1, 2, \ldots, m$. Such a system is *strong* if we additionally have $x_i \notin S_j$ for all $i \neq j$. The next theorem due to Füredi and Tuza appears in Jukna's textbook [4].

Theorem 2. *In any family of more than $\binom{r+s}{s}$ sets of cardinality at most r, at least $s+2$ of its members have a strong system of distinct representatives.*

Given an instance $(\mathcal{U} = \{1, \ldots, n\}, \mathcal{S} = \{S_1, \ldots, S_m\}, k)$ of Unique Coverage, put $r = k - 1$ and $s = k$ in the statement of the above theorem and we have a kernel of size $\binom{2k-1}{k-1} \leq \binom{2k}{k} \leq 2^{2k}$.

Corollary 3. Unique Coverage *admits a problem kernel of size* 4^k.

Note that this implies that there is an $O^*(4^{k^2})$ time FPT algorithm for the Unique Coverage problem.

For the case where each set of the input family has size at most b, for some constant b, there is a better kernel. By Theorem 2, if there exists at least $\binom{b+k}{k}$ sets in the input family, then there exists at least k sets with a strong system of distinct representatives.

Corollary 4. *If each set* $S \in \mathcal{S}$ *has size at most* b *then the* Unique Coverage *problem has a kernel of size* $O(2^{b+k})$.

4 The Budgeted Unique Coverage Problem

In this section we consider the Budgted Unique Coverage problem where each set in the input family has a cost and each element in the universe has a profit; the goal is to decide whether there exists a subcollection of total cost at most B that uniquely covers elements of total profit at least k. By parameterizing on k or B or both we obtain different parameterized versions of this decision question.

4.1 Hardness Results

We first consider the Budgeted Max Cut problem which is a specialization of the Budgeted Unique Coverage problem. An instance of this problem is an undirected graph $G = (V, E)$ with a cost function $c : V \to \mathbb{R}^+$ on the vertex set and a profit function $p : E \to \mathbb{R}^+$ on the edge set; positive real numbers B and k. The question is whether there exists a cut (T, T') such that the total cost of the vertices in T is at most B and the total profit of the edges crossing the cut is at least k.

We first show that the Budgeted Max Cut problem with arbitrary positive real costs and profits is probably not FPT.

Lemma 4. *The* Budgeted Max Cut *problem with arbitrary positive costs and profits with parameters* B *and* k *is not FPT, unless* $P = NP$.

Proof. Suppose there exists an algorithm for the Budgeted Max Cut problem (with arbitrary positive costs and profits) with running time $O(f(k, B) \cdot \text{poly}(n))$. We will use this to solve the decision version of Max Cut in polynomial time. Let $(G = (V, E), k)$ be an instance of the Max Cut problem, where $|V| = n$. Assign each vertex of the input graph cost $1/n$ and each edge profit $1/k$. Let the budget $B = 1/2$ and the profit $k' = 1$. Clearly, G has a maximum cut of size at least k iff there exists $S \subseteq V$ of total cost at most B such that the total profit of the edges crossing the cut $(S, V - S)$ is at least k'. And this can be answered in time $O(f(1, 1/2) \cdot p(|V|))$, implying $P = NP$. □

Theorem 3. *The* Budgeted Unique Coverage *problem with arbitrary positive costs and profits is not FPT, unless* $P = NP$.

Henceforth by the 'budgeted' version we mean the case when the costs and profits are positive integers. We next show that the Budgeted Max Cut problem parameterized by the budget B alone is $W[1]$-hard.

Lemma 5. *The* Budgeted Max Cut *problem parameterized by the budget is* $W[1]$*-hard.*

Proof. To show $W[1]$-hardness, we exhibit a fixed-parameter reduction from the Independent Set problem to the Budgeted Max Cut problem with unit costs and profits. Let $(G = (V, E), B)$ be an instance of Independent Set with $|V| = n$. For every vertex $u \in V$ add $|V| - 1 - \deg(u)$ new vertices and connect them to u. Call the resulting graph G'. Note that every vertex $u \in G$ has degree $|V| - 1$ in G'. We let $(G' = (V', E'), B, k = B(n-1))$ be the instance of Budgeted Max Cut.

Claim. G has an independent set of size B iff G' has a cut $(S, V' - S)$ such that $|S| = B$ and at least $k = B(n-1)$ edges lie across it.

If G has an independent set S of size B, then clearly S is independent in G'. The cut $(S, V' - S)$ does indeed have $B(n-1)$ edges crossing it, as every vertex of S has degree $n-1$. Next suppose that G' has a cut $(S, V' - S)$ such that $|S| = B$ and $B(n-1)$ edges cross the cut. Note that every vertex in S must be a vertex from G. Otherwise the cut cannot have $B(n-1)$ edges crossing it. Suppose two vertices u and v in S are adjacent. Then both u and v contribute less than $n-1$ edges to the cut. Since each vertex in S contributes at most $n-1$ edges to the cut, the number of edges crossing the cut must be less than $B(n-1)$, a contradiction. Hence S is independent in G' and hence G has an independent set of size B. □

Since the Budgeted Unique Coverage problem is a generalization of Budgeted Max Cut we have

Theorem 4. *The* Budgeted Unique Coverage *problem parameterized by the budget* B *is* $W[1]$*-hard.*

4.2 A Fixed-Parameter Tractability Result

In this subsection, we give an FPT algorithm for Budgeted Unique Coverage, when B and k are parameters, assuming that for every set S in the input family the number of sets with a non-empty intersection with S is at most some function of k. This is a natural situation in real-world applications; for example, in wireless networks. For the Budgeted Max Cut problem, for instance, every set is intersected by at most $k-1$ sets.

Let $(\mathcal{U} = \{1, \ldots, n\}, \mathcal{S} = \{S_1, \ldots, S_m\}, c, p, B, k)$ be an instance of the Budgeted Unique Coverage problem where $c : \mathcal{S} \to \mathbb{N}$ and $p : \mathcal{U} \to \mathbb{N}$. For $\mathcal{T} \subseteq \mathcal{S}$, define $c(\mathcal{T}) = \sum_{S \in \mathcal{T}} c(S)$ and $p(\mathcal{T})$ to be the total profit of the elements uniquely

covered by $\mathcal{T}$. If $S_i \in \mathcal{S}$, define $N[S_i]$ to be the set of all members of $\mathcal{S}$ that have a nonempty intersection with S_i. We can without loss of generality assume that $c(S_i) \leq B$ and $|S_i| \leq k-1$ for all $1 \leq i \leq m$. In what follows, we assume that for all $S_i \in \mathcal{S}$, we have $|N[S_i]| \leq f(k)$ for some function f.

The FPT algorithm that we describe here builds the solution in stages. Note that if we decide to include a set S in the solution, there is no way of deciding how many elements S covers uniquely unless we make choices for each set in $N[S]$. To get around this, the algorithm, at any stage, decides whether or not to include a subfamily $\mathcal{A} \subseteq N[S]$ for some set S. If it includes $\mathcal{A}$ in the solution, then it automatically *excludes* $N[S] \setminus \mathcal{A}$ from it. The current solution is a pair $(\mathcal{T}, \mathcal{T}')$, where $\mathcal{T}, \mathcal{T}' \subseteq \mathcal{S}$ and $\mathcal{T} \cap \mathcal{T}' = \emptyset$. The sets included by the algorithm in the solution till the current stage are in $\mathcal{T}$; those excluded from the solution are in $\mathcal{T}'$.

Call a pair $(\mathcal{T}, \mathcal{T}')$ a feasible solution for an instance of Budgeted Unique Coverage if $\mathcal{T}' = \mathcal{S} - \mathcal{T}$, $c(\mathcal{T}) \leq B$ and $p(\mathcal{T}) \geq k$. A pair $(\mathcal{T}, \mathcal{T}')$ is a *partial solution* if $\mathcal{T}, \mathcal{T}' \subseteq \mathcal{S}$ and $\mathcal{T} \cap \mathcal{T}' = \emptyset$. A partial solution $(\mathcal{T}, \mathcal{T}')$ can be *extended* to a feasible solution if there exist $\mathcal{X}, \mathcal{X}' \subseteq \mathcal{S} - (\mathcal{T} \cup \mathcal{T}')$ such that $\mathcal{X} \cap \mathcal{X}' = \emptyset$ and $(\mathcal{T} \cup \mathcal{X}, \mathcal{T}' \cup \mathcal{X}')$ is a feasible solution. A partial solution $(\mathcal{T}, \mathcal{T}')$ is *strong* if for each set $S_i \in \mathcal{T}$, $N[S_i] \subseteq \mathcal{T} \cup \mathcal{T}'$. Given a strong partial solution $(\mathcal{T}, \mathcal{T}')$, let $\mathcal{U}_1, \ldots, \mathcal{U}_t$ be a partition of $\mathcal{S} - (\mathcal{T} \cup \mathcal{T}')$ according to costs. That is, all members in any set $\mathcal{U}_i$ have the same cost c_i and for all $i \neq j$, $c_i \neq c_j$. Note that $t \leq B$. For each $\mathcal{U}_i$, let U_i^{max} denote a member of $\mathcal{U}_i$ with maximum total profit.

Lemma 6. *Let $(\mathcal{T}, \mathcal{T}')$ be a strong partial solution and let $\mathcal{U}_1, \mathcal{U}_2, \ldots, \mathcal{U}_t$ be a partition of $\mathcal{S} - (\mathcal{T} \cup \mathcal{T}')$ according to costs. Suppose $(\mathcal{T}, \mathcal{T}')$ can be extended to a feasible solution by adding a member of $\mathcal{U}_i$ to $\mathcal{T}$. Then there exists an extension of $(\mathcal{T}, \mathcal{T}')$ into a feasible solution such that $\mathcal{T} \cap N[U_i^{max}] \neq \emptyset$.*

Proof. Suppose $(\mathcal{T}, \mathcal{T}')$ can be extended to a feasible solution $(\mathcal{X}, \mathcal{X}')$ by adding a member $U \in \mathcal{U}_i$ to $\mathcal{T}$ and that $\mathcal{X} \cap N[U_i^{max}] = \emptyset$. This means $N[U_i^{max}] \subseteq \mathcal{X}'$. Remove U from $\mathcal{X}$ and replace it by U_i^{max}. Note that every element of U_i^{max} is uniquely covered and that the total profit of these newly uniquely covered elements is at least as that of those covered by U. Since $c(U) = c(U_i^{max})$, the new solution continues to be feasible. □

One can use Lemma 6 to develop an FPT algorithm with time complexity $O^*((B \cdot 2^{f(k)})^{B+k})$. Suppose there exists a feasible solution to the given input instance. We start with a strong feasible solution $(\mathcal{T} = \emptyset, \mathcal{T}' = \emptyset)$. Partition the input family $\mathcal{S}$ according to costs into the subfamilies $\mathcal{U}_1, \ldots, \mathcal{U}_t$. Note that $t \leq B$. Since there exists a feasible solution, it has to include a set from one of the subfamilies $\mathcal{U}_i$. For each choice of a subfamily, Lemma 6 assures us that it is sufficient to consider a set S in the subfamily which maximizes profit. We consider all possible bipartitions $(\mathcal{A}, \mathcal{A}')$ of $N[S]$ such that $\mathcal{A} \neq \emptyset$ and each member in $\mathcal{A}$ uniquely covers at least one element. For each such bipartition $(\mathcal{A}, \mathcal{A}')$, set $\mathcal{T} \leftarrow \mathcal{T} \cup \mathcal{A}$ and $\mathcal{T}' \leftarrow \mathcal{T}' \cup \mathcal{A}'$. Since by our assumption, $|N[S]| \leq f(k)$, there are at most $2^{f(k)}$ such bipartitions. This gives an initial branching factor of $B \cdot 2^{f(k)}$.

We then recurse using Lemma 6. In order to recurse, we must first ensure that the current partial solution is strong. We achieve this by considering all possible bipartitions of $N[T] - (\mathcal{T} \cup \mathcal{T}')$ for all sets $T \in \mathcal{T}$ for which $N[T] - (\mathcal{T} \cup \mathcal{T}') \neq \emptyset$. As before, we are interested in bipartitions $(\mathcal{A}, \mathcal{A}')$ which have the property that each set in $\mathcal{T} \cup \mathcal{A}$ uniquely covers at least one element. For each such bipartition $(\mathcal{A}, \mathcal{A}')$, we set $\mathcal{T} \leftarrow \mathcal{T} \cup \mathcal{A}$ and $\mathcal{T}' \leftarrow \mathcal{T}' \cup \mathcal{A}'$. There are at most 2^k such bipartitions and for each bipartition, we either increase the cost of the solution or total profit of uniquely covered elements by at least 1. If at any stage of recursion, we find that there is no subfamily $\mathcal{U}_i$ such that for $U \in \mathcal{U}_i$, $c(U) \leq B - c(\mathcal{T})$, we abort that branch. If $p(\mathcal{T}) \geq k$, at any stage, we halt and output YES. The overall depth of the recursion tree is at most $B + k$ and the branching factor is at most $B \cdot 2^{f(k)}$. The overall time complexity is therefore $O^*((B \cdot 2^{f(k)})^{B+k})$. If the algorithm does not return a solution then we can safely conclude that the given instance is a NO-instance.

Theorem 5. *Suppose $(\mathcal{U}, \mathcal{S}, c, p, B, k)$ is an instance of the* BUDGETED UNIQUE COVERAGE *problem where for every set $S \in \mathcal{S}$, we have $|N[S]| \leq f(k)$. Then there is an algorithm with time complexity $O^*((B \cdot 2^{f(k)})^{B+k})$ for this problem.*

The BUDGETED MAX CUT problem is a special case where $|N[S]| \leq k - 1$ for all $S \in \mathcal{S}$, and the following corollary is immediate.

Corollary 5. *The* BUDGETED MAX CUT *problem with positive integer costs and profits is fixed-parameter tractable when parameterized by B and k. There is an algorithm with time complexity $O^*((B \cdot 2^k)^{B+k})$ for this problem.*

5 Concluding Remarks

In this paper, we considered the parameterized complexity of the UNIQUE COVERAGE problem. There are several directions in which to proceed. Firstly, the reduction rules that we give are almost trivial and the kernel that we obtain is exponential in k. Kernelization is a very important topic in the design of FPT algorithms and the challenge is to devise reduction rules to obtain a polynomial (linear?) kernel or prove that no such kernel exists under some plausible complexity-theoretic assumption. Are there reduction rules that lead to a better problem kernel? In particular, is there a polynomial kernel for the UNIQUE COVERAGE problem?

At this point, all we can show is that with respect to a broader set of reduction rules, which we do not state here, the kernel size is at least $\Omega(2^k/\sqrt{k/2})$. The following example illustrates this situation. Let $\mathcal{U} = \{1, 2, \ldots, k\}$, $\mathcal{S} = \mathcal{S}_1 \cup \mathcal{S}_2$, where $\mathcal{S}_1$ consists of all subsets of $\mathcal{U}$ of size exactly $\lceil k/2 \rceil + 1$ and $\mathcal{S}_2$ is some collection of subsets of $\mathcal{U}$ of size at most $k/4$. Note that $|\mathcal{S}_1| = \binom{k}{\lceil k/2 \rceil + 1}$, which by Stirling's approximation is, $\Omega(2^k/\sqrt{k/2})$. If $\mathcal{S}_2 = \emptyset$ then one can show that the given instance is a NO-instance. But we can always produce an $\mathcal{S}_2 \neq \emptyset$ such that the given instance is a YES-instance and such that our reduction rules do not change the size of the input instance. For instance, if we take

$\mathcal{S}_2 = \{\{\lceil k/2 \rceil + 2\}, \ldots, \{k\}\}$, then this is a YES-instance and we can show that our reduction rules do not alter the size of the input.

Another important question is whether there exists a good branching algorithm for UNIQUE COVERAGE. The algorithm that we gave runs in time $O^*(4^{k^2})$. Finally, is the BUDGETED UNIQUE COVERAGE problem with positive integer costs/profits, with parameters B and k, fixed-parameter tractable?

Acknowledgements. We thank Saket Saurabh for pointing out the connection between UNIQUE COVERAGE and strong systems of distinct representatives.

References

1. Bar-Yehuda, R., Goldreich, O., Itai, A.: On the time-complexity of broadcast in multi-hop radio networks: An exponential gap between determinism and randomization. Journal of Computer and System Sciences 45, 104–126 (1992)
2. Demaine, E.D., Hajiaghayi, M.T., Feige, U., Salavatipour, M.R.: Combination can be hard: approximability of the unique coverage problem. In: Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2006), SIAM, pp. 162–171 (2006)
3. Downey, R.R., Fellows, M.R.: Parameterized Complexity. Springer, New York (1999)
4. Jukna, S.: Extremal combinatorics. Springer, Berlin (2001)
5. Niedermeier, R.: An Invitation to Fixed-Parameter Algorithms. Oxford University Press, Oxford (2006)
6. Paz, A., Moran, S.: Non deterministic polynomial optimization problems and their approximations. Theoretical Computer Science 15, 251–277 (1981)

Bounded Tree-Width and CSP-Related Problems

Tommy Färnqvist* and Peter Jonsson**

Department of Computer and Information Science
Linköpings Universitet
S-581 83 Linköping, Sweden
{tomfa,petej}@ida.liu.se

Abstract. We study the complexity of structurally restricted homomorphism and constraint satisfaction problems. For every class of relational structures $\mathcal{C}$, let $\text{LHOM}(\mathcal{C}, \_)$ be the problem of deciding whether a structure $\mathbf{A} \in \mathcal{C}$ has a homomorphism to a given arbitrary structure $\mathbf{B}$, when each element in $\mathbf{A}$ is only allowed a certain subset of elements of $\mathbf{B}$ as its image. We prove, under a certain complexity-theoretic assumption, that this *list homomorphism problem* is solvable in polynomial time if and only if all structures in $\mathcal{C}$ have bounded tree-width. The result is extended to the connected list homomorphism, edge list homomorphism, minimum cost homomorphism and maximum solution problems. We also show an inapproximability result for the minimum cost homomorphism problem.

Keywords: Computational complexity, constraint satisfaction, homomorphism, relational structure, inapproximability.

1 Introduction

A large class of problems in different areas of computer science can be viewed as constraint satisfaction problems [2,7,13,15,20,23]. This includes problems in artificial intelligence, database theory, scheduling, frequency assignment, graph theory and satisfiability. The main model [13] considers constraint satisfaction problems with a fixed template determining the size of the domain and the set of allowed constraint types in an instance. Feder and Vardi [13] observed that constraint satisfaction problems can be described as homomorphism problems for relational structures. For an excellent introduction to and survey of the strongly related subject of *graph* homomorphisms, we refer to [17]. For every two classes of relational structures $\mathcal{C}$, $\mathcal{D}$, let $\text{HOM}(\mathcal{C}, \mathcal{D})$ be the problem of deciding whether a structure $\mathbf{A} \in \mathcal{C}$ has a homomorphism to a given arbitrary structure $\mathbf{B} \in \mathcal{D}$. To simplify the notation, if either $\mathcal{C}$ or $\mathcal{D}$ is the class of all structures, we just use the placeholder '_'. Grohe [15] has studied so called *structural* restrictions, i.e. the question of how to restrict $\mathcal{C}$, so that $\text{HOM}(\mathcal{C}, \_)$ is polynomial-time solvable. He proves the following:

Assume that FPT $\neq$ W[1]. Then for every recursively enumerable class $\mathcal{C}$ of structures of bounded arity, $\text{HOM}(\mathcal{C}, \_)$ *is polynomial-time solvable if and only if the core of every structure in $\mathcal{C}$ has tree-width at most w (for some fixed w).*

* Supported by the *National Graduate School in Computer Science* (CUGS), Sweden.
** Partially supported by the *Center for Industrial Information Technology* (CENIIT) under grant 04.01, and by the *Swedish Research Council* (VR) under grant 2006-4532.

T. Tokuyama (Ed.): ISAAC 2007, LNCS 4835, pp. 632–643, 2007.

FPT $\neq$ W[1] is a standard assumption from parameterised complexity theory that is widely believed to be true. A *core* of a relational structure $\mathbf{A}$ is a substructure $\mathbf{A}' \subseteq \mathbf{A}$ such that there is a homomorphism from $\mathbf{A}$ to $\mathbf{A}'$, but no homomorphism from $\mathbf{A}'$ to a proper substructure of $\mathbf{A}'$. All cores of a structure $\mathbf{A}$ are isomorphic, so it is reasonable to speak of *the* core of $\mathbf{A}$.

In the *list homomorphism problem* [2,6,7,9,10,11,12], LHOM$(\mathcal{C}, \mathcal{D})$, the goal is to decide whether there is a homomorphism from a structure $\mathbf{A} \in \mathcal{C}$ to a given structure $\mathbf{B} \in \mathcal{D}$, when each element in $\mathbf{A}$ is only allowed a certain subset of elements in the universe of $\mathbf{B}$ as its image. Such list homomorphisms generalise e.g. list colourings and have many natural applications. We show the following:

Assume that FPT $\neq$ W[1]. Then for every recursively enumerable class $\mathcal{C}$ of structures of bounded arity, LHOM$(\mathcal{C}, _)$ *is polynomial-time solvable if and only if every structure in $\mathcal{C}$ has tree-width at most w (for some fixed w).*

Incidentally, this complexity-theoretic classification coincides with that of Dalmau and Jonsson's in [3], where they study the problem of counting homomorphisms. Our result is then extended to the *connected* list homomorphism problem [6], where every list has to induce a connected substructure of the right hand side input structure and to the *edge* list homomorphism problem [8], where the lists contain tuples from the relations of the right hand side input structure that the tuples on the left hand side have to map to. We remark that our hardness results still apply when the classes of relational structures are restricted to graphs. We also extend the result to two optimisation problems. The *minimum cost* homomorphism problem was introduced by Gutin et al. in [16], where it was motivated by a real-world problem in defence logistics. Here, mapping an element from the left hand side to an element on the right hand side is afflicted with costs and the objective is to find a homomorphism of minimum cost. This problem includes as special cases the list homomorphism problem and the general optimum cost chromatic partition problem [24]. In the *maximum solution problem* [21], the right hand side elements are assumed to be a finite subset of the natural numbers and the objective is to find a homomorphism that has maximum possible total weight. In some sense, see [21], this is a generalisation of integer programming and captures e.g. the INDEPENDENT SET problem. When the right hand side is restricted to $\{0, 1\}$ this is the well-studied MAX ONES problem. The hard instances of the minimum cost homomorphism problem are also shown to be inapproximable as well. To our knowledge, this is the first inapproximability result for this problem.

The rest of this paper is organised as follows. Section 2 introduces the requisite background material and problem definitions for several variants of the homomorphism problem. Section 3 contains proofs of our intractability and inapproximability results. Finally, Section 4 concludes the paper and presents possible future work.

2 Preliminaries

Most of the terminology presented in this section comes from [3,14,15,16]. In the next three subsections, we provide the necessary background material on relational structures and graph theory, homomorphism problems and parameterised complexity, respectively.

2.1 Relational Structures and Graph Theory

A *vocabulary* τ is a finite set of relation symbols of specified *arities*, denoted $ar(\cdot)$. The arity of τ is $\max\{ar(R) \mid R \in \tau\}$. A *$\tau$-structure* $\mathbf{A}$ consists of a finite set A (called the *universe* of $\mathbf{A}$) and for each relation symbol $R \in \tau$, a relation $R^A \subseteq A^{ar(R)}$. We say that a class $\mathcal{C}$ of structures is of *bounded arity* if there is an r such that every structure in $\mathcal{C}$ is at most r-ary. A *substructure* of a τ-structure $\mathbf{A}$ is a τ-structure $\mathbf{B}$ with universe $B \subseteq A$ and relations $R^B \subseteq R^A$ for all $R \in \tau$.

A substructure $\mathbf{B}$ is *induced* if for all $R \in \tau$, say, of arity r, $R^{\mathbf{B}} = R^{\mathbf{A}} \cap B^r$. We define the *size* $\|\mathbf{A}\|$ of the structure $\mathbf{A}$ as $\|\mathbf{A}\| = |\tau| + |A| + \sum_{R\in\tau} |R^{\mathbf{A}}| \cdot |ar(R)|$. $\|\mathbf{A}\|$ is roughly the size of a reasonable encoding of $\mathbf{A}$.

Let $\mathbf{A}$ and $\mathbf{B}$ be τ-structures. We define $\mathbf{A} \cup \mathbf{B}$ to be the τ-structure with universe $A \cup B$ and such that for all $R \in \tau$, $R^{\mathbf{A}\cup\mathbf{B}} = R^{\mathbf{A}} \cup R^{\mathbf{B}}$

Let E be a binary relation symbol. We view graphs as $\{E\}$-structures $\mathbf{G}$ and assume that they are undirected and loop-free. A graph $\mathbf{H}$ is a *minor* of a graph $\mathbf{G}$ if $\mathbf{H}$ is isomorphic to a graph that can be obtained from a subgraph of $\mathbf{G}$ by contracting edges. We define a *minor map* from $\mathbf{H}$ to $\mathbf{G}$ to be a mapping $\mu : H \to 2^G$ having the following properties:

1. for all $v \in H$, the set $\mu(v)$ is non-empty and connected in $\mathbf{G}$;
2. for all $v, w \in H$ with $v \neq w$, the sets $\mu(v)$ and $\mu(w)$ are disjoint; and
3. for all edges $\{v, w\} \in E^{\mathbf{H}}$, there are $v' \in \mu(v)$ and $w' \in \mu(w')$ such that $\{v', w'\} \in E^{\mathbf{G}}$.

We call a minor map μ from $\mathbf{H}$ to $\mathbf{G}$ *onto* if $\bigcup_{v\in H} \mu(v) = G$. It is easy to see that there is a minor map from $\mathbf{H}$ to $\mathbf{G}$ if and only if $\mathbf{H}$ is a minor of $\mathbf{G}$. Moreover, if $\mathbf{H}$ is a minor of a connected graph $\mathbf{G}$, then we can always find a minor map from $\mathbf{H}$ onto $\mathbf{G}$.

A *tree-decomposition* of a graph $\mathbf{G}$ is a pair $(\mathbf{T}, \beta)$ where $\mathbf{T}$ is a tree and $\beta : T \to 2^G$ satisfies the following conditions:

1. for every $v \in G$, the set $\{t \in T \mid v \in \beta(t)\}$ is non-empty and connected in $\mathbf{T}$; and
2. for every $e \in E^{\mathbf{G}}$, there is a $t \in T$ such that $e \subseteq \beta(t)$.

The width of a tree-decomposition $(\mathbf{T}, \beta)$ is $\max\{|\beta(t)| \mid t \in T\} - 1$, and the *tree-width* $\omega(\mathbf{G})$ of a graph $\mathbf{G}$ is the minimum w such that $\mathbf{G}$ has a tree-decomposition of width w.

For $k, \ell \geq 1$, the $(k \times \ell)$-grid is the graph with vertex set $\{1, \ldots, k\} \times \{1, \ldots, \ell\}$ and an edge between (i, j) and (i', j') if and only if $|i - i'| + |j - j'| = 1$. It is not hard to see that the $(k \times k)$-grid has tree-width k. Robertson and Seymour have proved the following theorem which is known as the Excluded Grid Theorem:

Theorem 1. *[25] For every k there exists a $w(k)$ such that the $(k \times k)$-grid is a minor of every graph of tree-width at least $w(k)$.*

We will now generalise some of the graph-theoretic notions defined above to arbitrary relational structures. The *Gaifman graph* of a τ-structure $\mathbf{A}$ is the graph $\mathbf{G}(\mathbf{A})$ with vertex set A and an edge between a and b if $a \neq b$ and there is a relation symbol $R \in \tau$, say, of arity r, and a tuple $(a_1, \ldots, a_r) \in R^A$ such that $a, b \in \{a_1, \ldots, a_r\}$. Henceforth,

we say that a subset $B \subseteq A$ is connected in a structure $\mathbf{A}$ if it is connected in $\mathbf{G}(\mathbf{A})$. A tree-decomposition of a τ-structure $\mathbf{A}$ is viewed as a tree-decomposition of $\mathbf{G}(\mathbf{A})$. A minor map from $\mathbf{A}$ to $\mathbf{B}$ is a mapping $\mu : A \to 2^B$ that is a minor map from $\mathbf{G}(\mathbf{A})$ to $\mathbf{G}(\mathbf{B})$.

2.2 Homomorphism Problems

A *homomorphism* from a τ-structure $\mathbf{A}$ to a τ-structure $\mathbf{B}$ is a mapping $h : A \to B$ such that for all $R \in \tau$, say, of arity r, and all tuples $(a_1, \ldots, a_r) \in R^A$, we have $(h(a_1, \ldots, h(a_r)) \in R^B$.

For two classes $\mathcal{C}$ and $\mathcal{D}$ of structures, $\text{HOM}(\mathcal{C}, \mathcal{D})$ is the following problem:

INSTANCE: $\mathbf{A} \in \mathcal{C}$, $\mathbf{B} \in \mathcal{D}$.
OUTPUT: "yes" if a homomorphism from $\mathbf{A}$ to $\mathbf{B}$ exists, "no" if no homomorphism from $\mathbf{A}$ to $\mathbf{B}$ exists.

If $\mathcal{D}$ is the class of all finite structures, we write $\text{HOM}(\mathcal{C}, \_)$ instead of $\text{HOM}(\mathcal{C}, \mathcal{D})$.

In the *list homomorphism problem*, each element of the left hand side input structure is given together with a set, called a *list*, of possible images in the right hand side input structure. This problem has been well studied with regard to restrictions to the right hand side input structure, see e.g. [2,6,7,9,10,11,12] for some results. We denote it $\text{LHOM}(\mathcal{C}, \mathcal{D})$:

INSTANCE: $\mathbf{A} \in \mathcal{C}$, $\mathbf{B} \in \mathcal{D}$, $L_a \subseteq B$ for each $a \in A$.
OUTPUT: "yes" if a homomorphism h from $\mathbf{A}$ to $\mathbf{B}$ such that $h(a) \in L_a$ for each $a \in A$ exists, "no" otherwise.

By restricting $\text{LHOM}(\mathcal{C}, \mathcal{D})$ to those inputs in which each list L_a induces a connected subgraph of the Gaifman graph $\mathbf{G}(\mathbf{B})$ of $\mathbf{B}$, we get the *connected list homomorphism problem*, $\text{CLHOM}(\mathcal{C}, \mathcal{D})$, introduced for graphs in [6]:

INSTANCE: $\mathbf{A} \in \mathcal{C}$, $\mathbf{B} \in \mathcal{D}$, $L_a \subseteq B$ for each $a \in A$, such that each L_a induces a connected substructure in $\mathbf{B}$.
OUTPUT: "yes" if a homomorphism h from $\mathbf{A}$ to $\mathbf{B}$ such that $h(a) \in L_a$ for each $a \in A$ exists, "no" otherwise.

Feder and Hell introduce the *edge list homomorphism problem* for undirected graphs in [8]. Here we generalise this to arbitrary relational structures and let $\text{ELHOM}(\mathcal{C}, \mathcal{D})$ be the following problem:

INSTANCE: $\mathbf{A} \in \mathcal{C}$, $\mathbf{B} \in \mathcal{D}$, lists of tuples from the relations of $\mathbf{B}$ for each tuple of the relations in $\mathbf{A}$.
OUTPUT: "yes" if a homomorphism h from $\mathbf{A}$ to $\mathbf{B}$ such that each tuple in the relations of $\mathbf{A}$ maps to a tuple in the corresponding list of tuples from $\mathbf{B}$ exists, "no" otherwise.

In [16] an optimisation problem is introduced, where every *graph* homomorphism is associated with a cost. We generalise this framework to arbitrary relational structures. If each element $a \in A$ is associated with, positive integral, costs $c_b(a)$, $b \in B$, then the cost of a homomorphism h is $\sum_{a \in A} c_{h(a)}(a)$ and the *minimum cost homomorphism problem*, $\text{MINHOM}(\mathcal{C}, \mathcal{D})$, is the following problem:

INSTANCE: $\mathbf{A} \in \mathcal{C}$, $\mathbf{B} \in \mathcal{D}$, positive integer costs $c_b(a)$, where $a \in A$ and $b \in B$.
OUTPUT: The cost of a minimum cost homomorphism from $\mathbf{A}$ to $\mathbf{B}$, "no" if no homomorphism from $\mathbf{A}$ to $\mathbf{B}$ exists.

If we let the universes B of the right hand side input structures of $\text{HOM}(\mathcal{C}, \mathcal{D})$ be finite subsets of the natural numbers equipped with the usual total order $<$, the *maximum solution problem* [21], $\text{MAX SOL}(\mathcal{C}, \mathcal{D})$, is the following problem:

INSTANCE: $\mathbf{A} \in \mathcal{C}$, $\mathbf{B} \in \mathcal{D}$, weight function $\omega : A \to \mathbb{N}$
OUTPUT: The maximum of $\sum_{a \in A} \omega(a) \cdot h(a)$ for any homomorphism h from $\mathbf{A}$ to $\mathbf{B}$, "no" if no homomorphism from $\mathbf{A}$ to $\mathbf{B}$ exists.

We note that MAX SOL is an extension of the MAX ONES problem and, as in [22], where Khanna et al. classify the approximability of MAX ONES with respect to restrictions to the right hand side input structure, we restrict our attention to instances of MAX SOL satisfying the following restriction: if a, a' occur in the same tuple $(a_1, \ldots, a_r)$ in some relation in $\mathbf{A}$, then $a \neq a'$ must hold. We say that a structure having this property is *replication free*.

2.3 Parameterised Complexity

Finally, we need some facts concerning parameterised complexity theory. Here we relax the classical notion of tractability, polynomial time computability, by admitting algorithms whose running time is exponential in some *parameter* of the problem instance that can be expected to be small in the typical application.

A parameterisation of a problem $P \subseteq \Sigma^*$ is a polynomial time computable mapping $\kappa : \Sigma^* \to \mathbb{N}$. If $(x, k) \in \Sigma^* \times \mathbb{N}$ is an instance of a parameterised decision problem, we call x the input and k the parameter. For example, the parameterised clique problem p-CLIQUE, is the following problem:

INPUT: graph $\mathbf{G}$.
PARAMETER: $k \in \mathbb{N}$.
OUTPUT: "Yes" if $\mathbf{G}$ has a clique of size k, "no" otherwise.

A parameterised problem (P, κ) over Σ is *fixed-parameter tractable* if there is a computable function $f : \mathbb{N} \to \mathbb{N}$, a constant $c \in \mathbb{N}$ and an algorithm that given $(x, k) \in \Sigma^* \times \mathbb{N}$ computes the solution in time $f(k) \cdot |x|^c$. FPT denotes the class of all fixed-parameter tractable parameterised problems.

An *fpt-reduction* from a parameterised problem (P, κ) over Σ to a parameterised problem (P', κ') over Σ' is a mapping $R : \Sigma^* \to (\Sigma')^*$ such that for all $x \in \Sigma^*$ we have $R(x) \in P'$, R is computable in time $f(\kappa(x)) \cdot |x|^c$ and $\kappa'(R(x)) \leq g(\kappa(x))$ (for computable functions $f, g : \mathbb{N} \to \mathbb{N}$ and a constant c).

Hardness and completeness of parameterised problems for a parameterised complexity class are defined in the usual way. Downey and Fellows [4] defined a hierarchy W[1] $\subseteq$ W[2] $\subseteq \cdots$ of parameterised complexity classes. They conjecture that this hierarchy is strict and that FPT is strictly contained in W[1]. p-CLIQUE is shown to be W[1]-complete under fpt-reductions in [5]. This theorem is used in our hardness proofs.

The problems we are interested in are the homomorphism problems defined in Subsection 2.2 parameterised by the size of the left hand side input structure. E.g. we have the following definition of the parameterised list homomorphism problem, p-$\text{LHOM}(\mathcal{C}, \mathcal{D})$:

INPUT: $\mathbf{A} \in \mathcal{C}$, $\mathbf{B} \in \mathcal{D}$, $L_a \subseteq B$ for each $a \in A$.
PARAMETER: $||\mathbf{A}||$.

OUTPUT: "yes" if a homomorphism h from $\mathbf{A}$ to $\mathbf{B}$ such that $h(a) \in L_a$ for each $a \in A$ exists, "no" otherwise.

The parameterised versions of the other problems in Subsection 2.2 are defined analogously and with the same parameter.

3 Main Results

We are now ready to prove the main results. First, we make the observation that when our homomorphism problems are restricted to classes of structures that have bounded tree-width, standard techniques using tree-decompositions, cf [17,19], may be employed to solve the problems in question in polynomial time. Then we see that what is left to do to get a classification of our problems, with regard to structural restrictions, is to prove hardness for classes of structures with unbounded tree-width. The proofs need a bit of preparation, that is taken care of in Subsection 3.1. Subsection 3.2 contains the actual proofs.

3.1 The Structure B

Let $\mathbf{A}$ be a connected τ-structure. Let $k \geq 2$, $K = \binom{k}{2}$, and $\mu : \{1,\ldots,k\} \times \{1,\ldots,K\} \to 2^A$ a minor map from the $(k \times K)$-grid onto $\mathbf{A}$. Let us assume that we have fixed some bijection ϱ between $\{1,\ldots,K\}$ and the set of all unordered pairs of elements of $\{1,\ldots,k\}$. For improved readability, we write $i \in p$ instead of $i \in \varrho(p)$.

Let the $\{E\}$-structure $\mathbf{G}$ be a graph. We now concentrate on the τ-structure $\mathbf{B} = \mathbf{B}(\mathbf{A}, \mu, \mathbf{G})$, as defined by Grohe [15]. The universe B of $\mathbf{B}$ is given by:

$$\{(v,e,i,p,a) |\ v \in G, e \in E^{\mathbf{G}}, \\ 1 \leq i \leq k, 1 \leq p \leq K \text{ s.t. } (v \in e \iff i \in p), \\ a \in \mu(i,p)\}$$

We define the function $\Pi : B \to A$ by letting $\Pi(v,e,i,p,a) = a$. As usual, we extend Π and Π^{-1} to tuples by defining it component-wise.

For every relation $R \in \tau$, say, of arity r, and for all tuples $(a_1,\ldots,a_r) \in R^A$, we add to R^B all tuples $(b_1,\ldots,b_r) \in \Pi^{-1}(a_1,\ldots,a_r)$ satisfying the following two constraints for all $b, b' \in \{b_1,\ldots,b_r\}$:

(C1) if $b = (v,e,i,p,a)$ and $b' = (v',e',i,p',a')$, then $v = v'$; and
(C2) if $b = (v,e,i,p,a)$ and $b' = (v',e',i',p,a')$, then $e = e'$.

In the remainder of this paper, we will focus on homomorphisms from $\mathbf{A}$ to $\mathbf{B}$ such that each $a \in A$ is mapped to an element $b \in B$ that was "generated" by a, i.e. $b \in \Pi^{-1}(a)$. We will denote this by saying that for a homomorphism $h : A \to B$, $h(a) = (\_,\_,\_,\_,a)$ for each $a \in A$, where the placeholders '_' are used to indicate that the values in question are arbitrary, as long as the element is a member of B. To proceed we need the following fact:

Lemma 2. *The graph* $\mathbf{G}$ *contains a* k*-clique if and only if there exists a homomorphism* h *from* $\mathbf{A}$ *to* $\mathbf{B}$ *such that* $h(a) = (\_,\_,\_,\_,a)$ *for all* $a \in A$.

Proof. In the proof of Lemma 3.1 in [3] it is shown that the graph $\mathbf{G}$ contains a k-clique if and only if there exists a homomorphism h from $\mathbf{A}$ to $\mathbf{B}$ satisfying $\Pi \circ h = \mathbf{id}$, where $\mathbf{id}$ is the identity function on the set A. Now, if h is a homomorphism from $\mathbf{A}$ to $\mathbf{B}$ such that $h(a) = (\_, \_, \_, \_, a)$ for all $a \in A$, h obviously satisfies $\Pi \circ h = \mathbf{id}$ and vice versa. □

3.2 Hardness Results

The problem p-LHOM$(\mathcal{C}, _)$ is trivially in FPT when LHOM$(\mathcal{C}, _)$ is in FP, and we know that LHOM$(\mathcal{C}, _)$ is solvable in polynomial time if the structures in $\mathcal{C}$ have bounded tree-width. What is left to prove, to achieve the result announced in Section 1, is that if p-LHOM$(\mathcal{C}, _)$ is in FPT, then the structures in $\mathcal{C}$ have bounded tree-width. We do this by assuming that p-LHOM$(\mathcal{C}, _)$ is in FPT even when $\mathcal{C}$ has unbounded tree-width and showing that this implies p-CLIQUE is in FPT, in contradiction with the fact that it is W[1]-complete. This is accomplished by exhibiting an fpt-reduction from p-CLIQUE to p-LHOM$(\mathcal{C}, _)$, where the result in the previous subsection is applied. As the same reasoning applies to the four other problems under study, this proof is then adapted and extended to fit our different problem variations. However, due to space constraints, some proofs are omitted from this paper.

Lemma 3. *Let $\mathcal{C}$ be a recursively enumerable class of structures of bounded arity that does not have bounded tree-width. If either p-*LHOM$(\mathcal{C}, _)$*, p-*CLHOM$(\mathcal{C}, _)$*, p-*ELHOM$(\mathcal{C}, _)$ *or p-*MINHOM$(\mathcal{C}, _)$ *is in FPT, then FPT $=$ W[1].*

Proof. Let $(\mathbf{G}, k)$ be an instance of p-CLIQUE. By the Excluded Grid Theorem, there is some structure $\mathbf{A}$ in $\mathcal{C}$ such that the $(k \times K)$-grid is a minor of the Gaifman graph of $\mathbf{A}$. We enumerate the recursively enumerable class $\mathcal{C}$ until we find such an $\mathbf{A} = \mathbf{A}(k)$. Then we compute a minor map μ from the $(k \times K)$-grid to $\mathbf{A}$. Let $\mathbf{A_1}, \ldots, \mathbf{A_m}$ be a decomposition of $\mathbf{A}$ into its connected components. We can assume, without loss of generality, that the $(k \times K)$-grid is a minor of (the Gaifman graph of) $\mathbf{A_1}$ and that the minor map μ is onto $\mathbf{A_1}$.

Let $\mathbf{B} = (\mathbf{A}, \mu, \mathbf{G})$ constructed as above. By Lemma 2, we know that in order to decide if there exists a k-clique in $\mathbf{G}$ we only need to check if there is a homomorphism h from $\mathbf{A_1}$ to $\mathbf{B}$ such that h maps every $a \in A_1$ to some $(\_, \_, \_, \_, a) \in B$, since such an h exists if and only if $\mathbf{G}$ has a k-clique. We would like to differentiate $\mathbf{B}$, so that only homomorphisms mapping $a \in A_1$ to $(\_, \_, \_, \_, a) \in B$ are allowed. Fortunately, the list homomorphism problem lets us enforce precisely such a differentiation of $\mathbf{B}$.

To do this construct $\mathbf{B}'$ as $\mathbf{B} \cup \mathbf{A_2} \cup \ldots \cup \mathbf{A_m}$ and lists $L_a \subseteq B'$, $a \in A$ defined by:

$$L_a = \begin{cases} \{b \mid b \in B \text{ and } b = (\_, \_, \_, \_, a)\} \text{ if } a \in A_1 \\ \{b \mid b \in B' \setminus B, b = a\} \text{ otherwise} \end{cases}$$

This way, we will always be able to find a homomorphism from $\mathbf{A} \setminus \mathbf{A_1}$ to $\mathbf{B}' \setminus \mathbf{B}$: it is just a matter of selecting the only element b available in L_a for each $a \in A \setminus A_1$. Since $b = a$ in each case this obviously results in a homomorphism from $\mathbf{A} \setminus \mathbf{A_1}$ to $\mathbf{B}' \setminus \mathbf{B}$.

It is also clear that the only possible homomorphisms h from $\mathbf{A_1}$ to $\mathbf{B}$ (and hence also the only possible homomorphisms from $\mathbf{A}$ to $\mathbf{B}'$), under our lists, are the ones

obeying the condition that h maps each $a \in A_1$ to some $(\_,\_,\_,\_,a) \in B$, due to the definition of the lists for elements $a \in A_1$.

Thus, the conclusion is that if $\mathbf{G}$ contains a k-clique, then we will be able to find a homomorphism from $\mathbf{A}$ to $\mathbf{B}'$, since then a homomorphism h from $\mathbf{A}_1$ to $\mathbf{B}$, obeying $h(a) = (\_,\_,\_,\_,a)$ for each $a \in A_1$, exists (by Lemma 2). If $\mathbf{G}$ has no k-clique, then we will not be able to find any homomorphism from $\mathbf{A}$ to $\mathbf{B}'$.

The construction of $\mathbf{A}$ only depends on k and is polynomial-time because $\mathcal{C}$ is recursively enumerable. Computing the minor map μ may require exponential time in the size of $\mathbf{A}$, but this is still bounded in terms of k. The size of an r-ary relation $R^{\mathbf{B}}$ is at most $|\Pi^{-1}(A^r)| \leq (|V^G| \cdot |E^G| \cdot |A|)^r$. This is polynomial in $||\mathbf{A}||$ and $||\mathbf{G}||$ since the arity of $\mathcal{C}$ is bounded. It follows that the size of $\mathbf{B}$ and $\mathbf{B}'$ is polynomially bounded in terms of $||\mathbf{A}||$ and $||\mathbf{G}||$ and so, $\mathbf{B}'$ can be computed in polynomial time. The lists L_a for $a \in A \setminus A_1$ are easy to compute and only hold one element each. While generating $\mathbf{B}$ it is easy to construct the lists L_a for $a \in A_1$ and the size of these lists are linear in the size of B. This shows that the reduction from $\mathbf{G}, k$ to $\mathbf{A}, \mathbf{B}', L_a$ is an fpt-reduction.

To be able to prove hardness for CLHOM we have to modify the structure $\mathbf{B}$ a bit; by adding some dummy elements to $\mathbf{B}$ we make our lists of elements in $\mathbf{B}$ induce connected substructures of $\mathbf{B}$. The result for ELHOM follows from a straightforward adaption of the proof for LHOM. Finally, the hardness result for MINHOM follows from transforming the instance of LHOM, in the proof of Lemma 3, to an instance of MINHOM by assigning $c_b(a) = 1$ if $b \in L_a$ and $c_b(a) = 2$ otherwise. □

An immediate consequence of the above is that the problem of *counting* list homomorphisms [18] is hard when $\mathcal{C}$ does not have bounded tree-width.

In the last reduction in the proof of Lemma 3, from p-CLIQUE to p-MINHOM$(\mathcal{C}, _)$, a gap that can be utilised to show the following (For further details regarding approximability we refer to [1].) is produced:

Proposition 4. *Let $\mathcal{C}$ be a recursively enumerable class of structures that does not have bounded tree-width. If* MINHOM$(\mathcal{C}, _)$ *is approximable within $2^{p(|A|)}$, (where p is a fixed polynomial), for every structure $\mathbf{A} \in \mathcal{C}$, then FPT $=$ W[1].*

Before we continue dealing with our hardness results, a remark about our chosen proof method is in place. Why do we need to use the structure $\mathbf{B}$ at all, could we not just reduce LHOM$(\mathcal{C}, _)$ to HOM$(\mathcal{C}', _)$, for some suitable class $\mathcal{C}'$, i.e. for $\mathbf{A} \in \mathcal{C}$, let $\mathbf{A}' \in \mathcal{C}'$ be the expansion of $\mathbf{A}$ having a relation R_a for each $a \in A$ such that $R_a^{A'} = \{(a)\}$, and go from there? This way, an instance $(\mathbf{A}, \mathbf{B})$ of LHOM reduces to $(\mathbf{A}', \mathbf{B}')$, where $\mathbf{B}'$ has $R_a^{B'} = L_a$ and $R^{B'} = R^B$ for all other relations R. The error in this line of reasoning, is that the structure $\mathbf{A}$ might not allow unary relations on all its members. To illustrate this point, think of the problems HOM$(\mathcal{C}, _)$ for a class $\mathcal{C}$ of structures with unbounded tree-width. Using the method of adding unary relations to the structures in $\mathcal{C}$, described above, we can now modify the proof of Lemma 3 to become a hardness proof for HOM$(\mathcal{C}, _)$! This is, of course, contradictory to Grohe's result. (If $\mathcal{C}$ is *restricted* to classes of structures that have all unary singleton relations, the cores of the structures can not be smaller than the structures themselves and the tree-widths of the structures and their respective cores coincide.)

In the hardness proof for MAX SOL, we are able to exploit the fact that we have to impose some total order on the elements in B; by letting elements of the form $(\_,\_,\_,\_,a)$, for some $a \in A$, have essentially the same values and inter-spacing these clusters with large gaps, the positive and negative instances of p-CLIQUE are separated.

Lemma 5. *Let $\mathcal{C}$ be a recursively enumerable class of replication free structures of bounded arity that does not have bounded tree-width. If p-MAX SOL$(\mathcal{C},_)$ is in FPT, then FPT $=$ W[1].*

Proof. We start out as in the proof of Lemma 3 and construct $\mathbf{B}'$ as $\mathbf{B} \cup \mathbf{A_2} \cup \ldots \cup \mathbf{A_m}$. To proceed, we have to impose some total order on the elements in B'. Fix the natural order $<$ on $\mathbb{N}$. The intuition is to let elements in B on the form $(\_,\_,\_,\_,a)$, for some $a \in A_1$, have essentially the same values in B'. If these small intervals where the $(\_,\_,\_,\_,a) \in B$ reside, for each a, are inter-spaced by large gaps and the weights assigned to $a \in A_1$ are chosen accordingly we might be able to separate the positive and negative instances of p-CLIQUE.

Let $\sigma = \max_{a \in A_1} |\Pi^{-1}(a)|$, the maximum number of elements in B "generated" by an element in A_1. Clearly, σ is bounded in terms of k and $||\mathbf{G}||$.

Let $B' \setminus B = \{1, \ldots, d\}$. Also, let $w(a) = 0$ when $a \in A \setminus A_1$. Furthermore, take an $a \in A_1$, let $w(a) = d + 1$ and let each $b \in \Pi^{-1}(a)$ have a distinct value in $[d+1, d+\sigma]$. The next $a \in A_1$ gets $w(a) = d + \Delta + 1$ while the associated $b \in \Pi^{-1}(a)$ get distinct values in $[d+\Delta+1, d+\Delta+\sigma]$. We continue this process until A_1 is exhausted and end up with the arrangement in Figure 1.

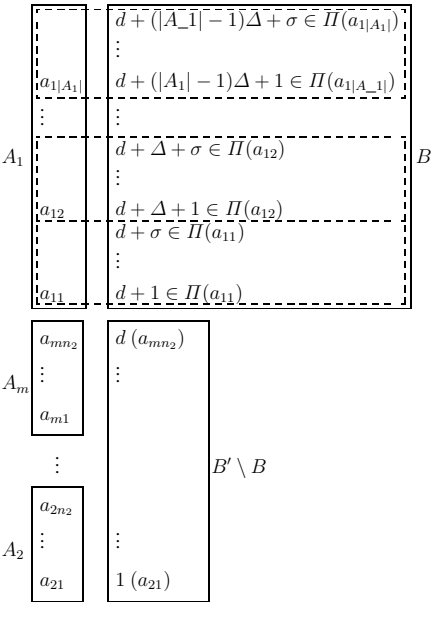


Fig. 1. The total order imposed on B'

We are interested in homomorphisms h between $\mathbf{A_1}$ and $\mathbf{B}$, such that each $a \in A_1$ maps to some $(\_,\_,\_,\_,a) \in B$, i.e. where the $a \in A_1$ with highest weight get mapped to some $(\_,\_,\_,\_,a) \in B$ in the highest interval of values, the $a \in A_1$ with second highest weight get mapped to some $(\_,\_,\_,\_,a) \in B$ in the second highest interval of values and so on. Such an h will receive a measure m_{id} with

$$(d+1)^2 + (d+\Delta+1)^2 + \ldots + (d+(|A_1|-1)\Delta+1)^2 \leq m_{id} \leq$$
$$\leq (d+1)(d+\sigma) + (d+\Delta+1)(d+\Delta+\sigma) + \ldots +$$
$$+(d+(|A_1|-1)\Delta+1)(d+(|A_1|-1)\Delta+\sigma).$$

It is easy to extend h to a homomorphism h' from $\mathbf{A}$ to $\mathbf{B}'$ (by mapping each $a \in A \setminus A_1$ to the $b \in B' \setminus B$ with $b = a$) and the measure for h' will still be m_{id}.

What false positives could we get? Recall that for each relation $R \in \tau$ and for all tuples $(a_1,\ldots,a_r) \in R^{\mathbf{A}_1}$, we add tuples $(b_1,\ldots,b_r) \in \Pi^{-1}(a_1,\ldots,a_r)$ satisfying certain conditions to $R^{\mathbf{B}}$ and that, in this case, $\mathbf{A}_1$ is replication free. This means that $\mathbf{B}$ is constructed so that any homomorphism h from $\mathbf{A}_1$ to $\mathbf{B}$ must have the property that the image of h contains at most one element from each interval, $[d+n\Delta+1, d+n\Delta+\sigma]$, of values in B.

That leaves the possibility that some intervals of values have been permuted in some way, i.e. at least a pair of elements in A_1 have been mapped to somewhere in "each others" intervals. It can be shown by induction that the maximum measure of such a homomorphism occurs when the two elements in A_1 that have lowest weight have swapped intervals, i.e. we have $h(a_{11}) = (\_,\_,\_,\_,a_{12})$ and $h(a_{12}) = (\_,\_,\_,\_,a_{11})$ in Figure 1, and the maximum value of each interval is picked as image. This measure matches the maximum possible m_{id} except for the two first summands.

The difference, denoted δ, between the lowest possible m_{id} and the measure of such a homomorphism is

$$\delta = \sum_{n=1}^{|A_1|}(d+(n-1)\Delta+1)^2 - (d+1)(d+\Delta+\sigma) - (d+\Delta+1)(d+\sigma) -$$
$$-\sum_{n=3}^{|A_1|}(d+(n-1)\Delta+1)(d+(n-1)\Delta+\sigma),$$

which is the same as (omitting the calculations) δ being equal to

$$\Delta^2 + \left(|A_1|^2 - |A_1| - \sigma|A_1|^2 + \sigma|A_1|\right)\Delta/2 + |A_1| + d|A_1| - d\sigma|A_1| - \sigma|A_1|.$$

If we choose Δ large enough, for example $\Delta = d^2\sigma^2|A_1|^2$, the difference δ will be positive and hence, we can say that if we find a homomorphism with measure m_{id}, $\mathbf{G}$ has a k-clique and that if the maximum measure of any homomorphism from $\mathbf{A}$ to $\mathbf{B}'$ is strictly less than the smallest possible m_{id}, $\mathbf{G}$ contains no k-clique. □

4 Conclusions and Open Questions

We have utilised the structure $\mathbf{B}$ defined by Grohe to classify a number of homomorphism problems by computational complexity with regard to structural restrictions, under the assumption that FPT $\neq$ W[1]. It is interesting to note that while the variants of

the homomorphism problem we have treated have their boundary between tractability and intractability at bounded tree-width of the left hand side input structure, the original $\text{HOM}(\mathcal{C}, \_)$ problem exhibits the same boundary at bounded tree-width for the core of the structures in $\mathcal{C}$. It would be interesting to characterise exactly what properties make the computational complexity of our problems different from that of the "regular" homomorphism problem.

Of course it would be nice to classify further homomorphism problems. E.g. the *retraction problem*, also known as the *one-or-all list homomorphism problem*, see [6], would be an interesting subject. Here, inputs of the list homomorphism problem are restricted to each list containing only a single element or the entire universe of the right hand side input structure.

In the reduction from p-CLIQUE to p-$\text{MINHOM}(\mathcal{C}, \_)$ a gap that can be used to show inapproximability properties of the intractable instances is produced. A gap is also produced in the MAX SOL case, but it is not exploitable in the same way. Is it possible to change the reduction somewhat to achieve a gap large enough for proving inapproximability?

A further observation is that the structure $\mathbf{B}$, so far, only has been applied when classifying homomorphism problems: is it possible to modify the structure $\mathbf{B}$, or the analysis of it, so that hardness proofs for problems where the solution is not necessarily a homomorphism, e.g. MAX CSP, becomes plausible?

Acknowledgements

We appreciate the comments of an anonymous reviewer who helped improve the presentation of this paper.

References

1. Ausiello, G., Crescenzi, P., Gambosi, G., Kann, V., Marchetti, A.: Complexity and approximation: Combinatorial optimization problems and their approximability properties. Springer, Heidelberg (1999)
2. Bulatov, A.A.: Tractable conservative constraint satisfaction problems. ACM Transactions on Computational Logic (to appear)
3. Dalmau, V., Jonsson, P.: The complexity of counting homomorphisms seen from the other side. Theoretical Computer Science 329(1-3), 315–323 (2004)
4. Downey, R.G., Fellows, M.R.: Fixed-parameter tractability and completeness I: Basic results. SIAM Journal on Computing 24, 873–921 (1995)
5. Downey, R.G., Fellows, M.R.: Fixed-parameter tractability and completeness II: On completeness for W[1]. Theoretical Computer Science 141, 109–131 (1995)
6. Feder, T., Hell, P.: List homomorphisms to reflexive graphs. J. Comb. Theory Series B 72, 236–250 (1998)
7. Feder, T., Hell, P.: Full constraint satisfaction problems. SIAM Journal on Computing 36, 230–246 (2006)
8. Feder, T., Hell, P.: Edge list homomorphisms, manuscript
9. Feder, T., Hell, P., Huang, J.: List homomorphisms and circular arc graphs. Combinatorica 19, 487–505 (1999)

10. Feder, T., Hell, P., Huang, J.: Bi-arc graphs and the complexity of list homomorphisms. J. Graph Theory 42, 61–80 (1999)
11. Feder, T., Hell, P., Huang, J.: List homomorphisms to reflexive digraphs, manuscript (2005)
12. Feder, T., Hell, P., Huang, J.: List homomorphisms of graphs with bounded degree. in Discrete Math (to appear)
13. Feder, T., Vardi, M.Y.: The computational structure of monotone monadic SNP and constraint satisfaction: a study through datalog and group theory. SIAM Journal on Computing 28(1), 57–104 (1998)
14. Flum, J., Grohe, M.: The parameterized complexity of counting problems. In: Proceedings of the 43rd IEEE Symposium on Foundations of Computer Science, pp. 538–547 (2002)
15. Grohe, M.: The complexity of homomorphism and constraint satisfaction problems seen from the other side. Journal of the ACM 54(1), 1–24 (2007)
16. Gutin, G., Rafiey, A., Yeo, A., Tso, M.: Level of repair analysis and minimum cost homomorphisms of graphs. Discrete Appl. Math. 154, 881–889 (2006)
17. Hell, P.: Algorithmic aspects of graph homomorphisms. In: Survey in combinatorics 2003. London Math. Society Lecture Note Series, vol. 307, pp. 239–276. Cambridge University Press, Cambridge (2003)
18. Hell, P., Nešetřil, J.: Counting list homomorphisms for graphs with bounded degree. Graphs, morphisms and statistical physics, DIMACS Series in Discrete Mathematics and Theoretical Computer Science 63, 105–112 (2004)
19. Hicks, I., Koster, A., Kolotoğlu, E.: Branch and tree decomposition techniques for discrete optimization. In: TutORials 2005. INFORMS TutORials in Operations Research Series, pp. 1–29 (2005)
20. Jeavons, P.: On the algebraic structure of combinatorial problems. Theoretical Computer Science 200(1–2), 185–204 (1998)
21. Jonsson, P., Nordh, G.: Generalised integer programming based on logically defined relations. In: Proceedings of the 31st International Symposium on Mathematical Foundations of Computer Science, pp. 549–560 (2006)
22. Khanna, S., Sudan, M., Trevisan, L., Williamson, D.P.: The approximability of constraint satisfaction problems. SIAM Journal on Computing 30(6), 1863–1920 (2001)
23. Kolaitis, P.G., Vardi, M.: Conjunctive-query containment and constraint satisfaction. Journal of Computer and System Sciences 61, 302–332 (2000)
24. Kroon, L.G., Sen, A., Deng, H., Roy, A.: The optimal cost chromatic partition problem for trees and interval graphs. In: Graph-Theoretic Concepts in Computer Science, pp. 279–292 (1997)
25. Robertson, N., Seymour, P.: Graph minors V: Excluding a planar graph. Journal of Combinatorial Theory, ser. B 62, 323–348 (1994)

Covering Points by Unit Disks of Fixed Location

Paz Carmi[1], Matthew J. Katz[2], and Nissan Lev-Tov[2]

[1] School of Computer Science, Carleton University, Ottawa, Canada
paz@cg.scs.carleton.ca
[2] Department of Computer Science, Ben-Gurion University, Beer-Sheva, Israel
{matya,nissanl}@cs.bgu.ac.il

Abstract. Given a set $\mathcal{P}$ of points in the plane, and a set $\mathcal{D}$ of unit disks of fixed location, the *discrete unit disk cover* problem is to find a minimum-cardinality subset $\mathcal{D}' \subseteq \mathcal{D}$ that covers all points of $\mathcal{P}$. This problem is a geometric version of the general set cover problem, where the sets are defined by a collection of unit disks. It is still NP-hard, but while the general set cover problem is not approximable within $c \log |\mathcal{P}|$, for some constant c, the discrete unit disk cover problem was shown to admit a constant-factor approximation. Due to its many important applications, e.g., in wireless network design, much effort has been invested in trying to reduce the constant of approximation of the discrete unit disk cover problem. In this paper we significantly improve the best known constant from 72 to 38, using a novel approach. Our solution is based on a 4-approximation that we devise for the subproblem where the points of $\mathcal{P}$ are located below a line l and contained in the subset of disks of $\mathcal{D}$ centered above l. This problem is of independent interest.

1 Introduction

We consider the problem of covering a given set of points in the plane by a given set of unit disks. Formally, we are given a set of points $\mathcal{P}$ in the plane, and a set of disks $\mathcal{D} = \{D_1, D_2, ..., D_n\}$ of radius 1 and centers $\mathcal{O} = \{o_1, o_2, ..., o_n\}$. We would like to find a minimum-cardinality subset $\mathcal{D}' \subseteq \mathcal{D}$, such that for each point $p \in \mathcal{P}$ there exists a disk $D \in \mathcal{D}'$ that contains p. We call this problem *discrete unit disk cover*.

The discrete unit disk cover problem (DUDC) has numerous applications, in particular in wireless network design. We are given a set of potential planar locations for placing base stations, and a set of points in the plane representing static clients. All wireless base stations have the same transmission range, where a client can "hear" the signals of a base station if and only if he/she is located within a disk of radius 1 around the base station. We are required to choose a minimum set of base stations such that each client is served by one or more base stations of the chosen set.

The problem of covering a given set of points by unit disks where the disk center locations are not restricted to a given set of points but rather may be chosen at any point in the plane, is studied in [4,5]. A polynomial-time approximation scheme (PTAS) is given for this problem using a grid-shifting strategy.

T. Tokuyama (Ed.): ISAAC 2007, LNCS 4835, pp. 644–655, 2007.

The discrete case, studied in this paper, where the center locations are restricted to a given set, is harder to approach since in order to cover the points within a constant size square one might need more than a constant number of the given disks.

This problem is a geometric set cover problem, where the given sets are defined by unit disks. It is still NP-hard [6]. However, this geometric restriction on the sets allows us to achieve a constant factor approximation, while the general set cover problem is not approximable within $c \log |\mathcal{P}|$, for some constant c, [9]. Due to the importance of the discrete unit disk cover problem, a continuous attempt has been made to achieve a constant approximation algorithm with a good constant factor. Brönnimann and Goodrich [1] gave an ϵ-net based algorithm where the constant factor is not specified. A 108-approximation for the discrete unit disk cover problem was presented in [2]. Narayanappa and Vojtechovsky [8] later improved this constant to 72 and stated that this is the best constant that can be achieved using their technique. In this paper we show that this constant can be reduced to 38 using a new approach.

Our algorithm is based on the single line problem, in which there exists a separating line such that the points to be covered are all located on one side of the line and contained in unit disks centered on the other side of the line. The covering disks may be chosen from both sides of the line. We present a 4-approximation algorithm for this special case and use this solution for approximating the general case. We partition the plane by a grid of width 3/2 and apply the 4-approximation twice for each grid line (once for each direction). We then consider each grid cell separately in order to take care of the uncovered points.

2 A 4-Approximation for the Single Line Problem

2.1 Setting

Let l be a horizontal line. Let $\mathcal{U}$ denote the disks of $\mathcal{D}$ centered above l and let $\mathcal{L} = \mathcal{D} \setminus \mathcal{U}$. We first provide some notation for the arrangement formed by the disks of $\mathcal{U}$ below l. Let B denote the region below l covered by the disks of $\mathcal{U}$. A disk $D \in \mathcal{U}$ is called a *lower boundary disk* if it contributes an arc to the boundary of B, or equivalently, if there exists a point $p \in D \cap B$ that does not belong to any other disk. (Otherwise it is called a *non-boundary disk*.) We then call the region $D \cap B$ a *lower boundary segment* and the arc $circ(D) \cap B$ a *lower boundary arc* (see Figure 1).

Let $\mathcal{S}$ be the set of all lower boundary segments of $\mathcal{U}$. Consider the arrangement $\texttt{Cells}(\mathcal{S})$ formed by the segments in $\mathcal{S}$. Assume the boundary disks are indexed according to their left intersection point with l, and associate with each cell of $\texttt{Cells}(\mathcal{S})$ the set of the indices of the segments that contain it. The next lemma (whose proof is omitted for lack of space) states that for each cell in $\texttt{Cells}(\mathcal{S})$, the set of indices associated with it forms a consecutive set of indices $i, i+1, \ldots, j$ for some $i \leq j$. We call such a cell an *interval cell* and denote it by $\texttt{icell}(i, j)$. We then say that $\mathcal{S}$ forms a *semi-chain*.

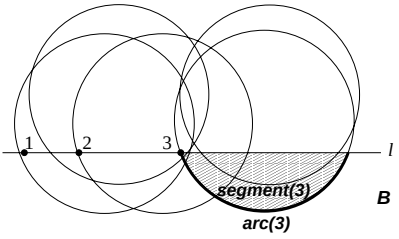


Fig. 1. Segments and arcs

Lemma 1. *Each cell of* $\texttt{Cells}(\mathcal{S})$ *is an interval cell.*

Consider the basic problem of covering the points of $\mathcal{P}$ that belong to B using only lower boundary disks. The following observation implies that the restriction to the set of lower boundary disks (rather than to the set $\mathcal{U}$) increases the size of the solution by a factor of at most 2.

Observation 1. *For any non-boundary segment s, there exist two consecutive boundary disks D_i, D_{i+1} that completely cover s. I.e., $\mathcal{P} \cap s \subseteq D_i \cup D_{i+1}$.*

Proof. Take the disk D_i to be the boundary disk that appears immediately before s (according to the left intersection point with the line l). □

An interval cell is said to be *occupied* if it contains a point of $\mathcal{P}$. For an interval $[i,j] \subseteq [1,k]$ let $\texttt{Occ}(i,j)$ denote the set of points of $\mathcal{P}$ contained in the occupied cells that correspond to subintervals of $[i,j]$. The following greedy algorithm finds a cover $\mathcal{C}$ of the points of $\mathcal{P}$ that belong to B, using a minimum subset of boundary disks. Initially set $\mathcal{C} = \emptyset$. At each step of the algorithm let i be the largest index such that all the points of $\texttt{Occ}(1,i)$ are covered by the disks in $\mathcal{C} \cup D_i$, and add D_i to $\mathcal{C}$. It is straightforward to see that the cover $\mathcal{C}$ is indeed of minimum cardinality if the covering set must consist of boundary disks.

2.2 Assisted Covers

Let $\mathcal{S}$ be the semi-chain formed by the lower boundary segments. Consider a disk $\tilde{D}$ centered below l, that intersects B. An interval $[i,j] \subseteq [1,k]$ with $i < j$ is said to be *assisted* by $\tilde{D}$ if the set $\{D_i, D_j, \tilde{D}\}$ covers all the points in $\texttt{Occ}(i,j)$. We then say that $\{D_i, D_j, \tilde{D}\}$ is an *assisting set* for $[i,j]$. (For $j = i+1$, we take the assisting set of $[i,j]$ to be $\{D_i, D_j\}$). A *left assisting pair* of an interval $[i,j]$ with $i < j$ is a pair $\{D_i, \tilde{D}\}$ where $\{D_i, D_j, \tilde{D}\}$ forms an assisting set for $[i,j]$. (For $j = i+1$ or $i = k$ (the last disk), we take the left assisting pair of $[i,j]$ to be D_i, that is, each chain disk itself is considered to be a left assisting pair.) Given the semi-chain $\mathcal{S}$, an *assisted cover* for $\mathcal{S}$ is a family $\mathcal{F}$ of left assisting pairs that covers all the points of $\mathcal{P}$ contained in $\texttt{Cells}(\mathcal{S})$.

In the example shown in Figure 2, the intervals $[1,3]$ and $[2,4]$ are assisted by $\tilde{D}$. The assisting sets are $\{D_1, D_3, \tilde{D}\}$, $\{D_1, D_2\}$, $\{D_2, D_3\}$, $\{D_2, D_4, \tilde{D}\}$ and

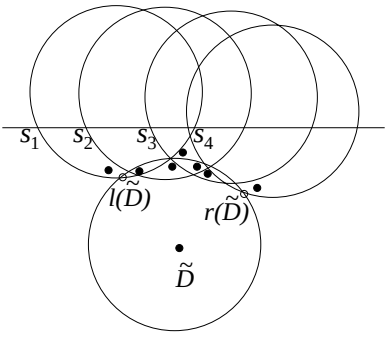


Fig. 2. $\tilde{D}$ assists the intervals $[1, 3]$ and $[2, 4]$

$\{D_3, D_4\}$. The left assisting pairs are $\{D_1, \tilde{D}\}$, $\{D_1\}$, $\{D_2\}$, $\{D_2, \tilde{D}\}$, $\{D_3\}$ and $\{D_4\}$. The family $\mathcal{F} = \{\{D_1\}, \{D_2, \tilde{D}\}, \{D_4\}\}$ forms an assisted cover.

Let $\mathcal{D}^*$ denote a minimum-cardinality (regular) cover of the points of $\mathcal{P}$ contained in B. Note that $\mathcal{D}^*$ can make use of all the disks in $\mathcal{D}$. Put $d^* = |\mathcal{D}^*|$. We now define the *minimum assisted cover* problem for $\mathcal{S}$, and show that a solution to this problem approximates d^*.

Minimum Assisted Cover Problem: Given a semi-chain $\mathcal{S}$, find a minimum-cardinality assisted cover $\mathcal{F}$ for $\mathcal{S}$.

For this problem we have the following lemma.

Lemma 2. *The minimum assisted cover problem has a polynomial-time solution.*

Proof. Given the semi-chain $\mathcal{S}$, the solution is constructed via an immediate reduction to the *minimum half-open interval cover* problem, defined as follows. Given a set $\mathcal{I}$ of points on the real line, and a family $\mathcal{J}$ of half-open intervals of the form $[a, b)$ (where a and b are both integers), find a minimum-cardinality family $\mathcal{J}' \subseteq \mathcal{J}$ that covers the points of $\mathcal{I}$ (assuming such a cover exists).

This one-dimensional problem can be solved easily using a greedy algorithm. The reduction is constructed as follows. For each point $p \in \mathcal{P} \cap B$ consider the largest index i such that s_i contains p. Let $\mathcal{I}$ be the set of indices corresponding to the points of $\mathcal{P} \cap B$. The family $\mathcal{J}$ consists of all half-open intervals $[i, j)$ such that for some $\tilde{D}$, $\{D_i, \tilde{D}\}$ is a left assisting pair for the interval $[i, j]$, including the half open intervals of the form $[i, i + 1)$ (and the interval $[k]$). □

Let Single Line denote the procedure that solves the minimum assisted cover problem for the semi-chain $\mathcal{S}$ using the reduction defined above, obtaining a family $\mathcal{F}$ of left assisting pairs. Partition the disks participating in $\mathcal{F}$ into two sets: the set $\mathcal{U}'$ of disks that belong to $\mathcal{S}$ (centered above l) and the set $\mathcal{L}'$ of assisting disks. Clearly $\mathcal{U}' \cup \mathcal{L}'$ is a cover of the points of $\mathcal{P}$ contained in B. Our goal is to show that $|\mathcal{U}' \cup \mathcal{L}'| \leq 4d^*$.

Let us now analyze the sizes of the sets $\mathcal{U}'$ and $\mathcal{L}'$. We have the following lemmas.

Lemma 3. *The family $\mathcal{F}$ and the set $\mathcal{U}'$ obtained by invoking Procedure* Single Line *satisfy $|\mathcal{F}| = |\mathcal{U}'|$.*

Proof. This holds since the chain disks of the left assisting pairs in $\mathcal{F}$ are distinct. □

Lemma 4. *The sets $\mathcal{U}'$ and $\mathcal{L}'$ obtained by invoking Procedure* Single Line*, satisfy $|\mathcal{L}'| \leq |\mathcal{U}'|$.*

Proof. Follows from the definition of left assisting pairs; for each assisting disk taken into $\mathcal{L}'$, at least one additional disk is taken into $\mathcal{U}'$. □

Consider an assisting disk $\tilde{D}$. Let $l(\tilde{D})$ (respectively, $r(\tilde{D})$), denote the leftmost (respectively, rightmost) point at which $\tilde{D}$ intersects the boundary of $\mathcal{S}$. Let $left(\tilde{D})$ denote the index i such that D_i contains $l(\tilde{D})$. Let $right(\tilde{D})$ denote the index j such that D_j contains $r(\tilde{D})$. We refer to the disk $D_{left(\tilde{D})}$ (respectively, $D_{right(\tilde{D})}$) as the *left bounding disk* (respectively, *right bounding disk*) of $\tilde{D}$. In Figure 2, $left(\tilde{D}) = 1$ and $right(\tilde{D}) = 4$.

For two assisting disks $\tilde{D}$ and $\tilde{D}'$, we say that $\tilde{D}$ is *dominated* by $\tilde{D}'$ with respect to the interval $[i, j]$ if all points in $\texttt{Occ}(i, j)$ that are covered by $\tilde{D}$ are also covered by $\tilde{D}'$. An assisting disk $\tilde{D}$ is called *strong assisting* if its center point $o(\tilde{D})$ lies above at least one of the points $l(\tilde{D})$ or $r(\tilde{D})$ (defined above). Otherwise $\tilde{D}$ is called *weak assisting*. For an assisting disk $\tilde{D}$, let the *left-right arc* of $\tilde{D}$ denote the upper part of $circ(\tilde{D})$ enclosed between $l(\tilde{D})$ and $r(\tilde{D})$.

We now have the following observations.

Observation 2. *Consider an interval $[i, j]$ and two weak assisting disks $\tilde{D}$ and $\tilde{D}'$ such that $left(\tilde{D}) \leq i$ and $left(\tilde{D}') \leq i$. Then either $circ(\tilde{D})$ and $circ(\tilde{D}')$ intersect each other exactly once in $\texttt{Cells}(i + 1, j - 1)$, or there is a dominance relationship between $\tilde{D}$ and $\tilde{D}'$ with respect to $[i + 1, j - 1]$.*

Proof. By the definition of weak assisting disks, the left-right arcs of $\tilde{D}$ and $\tilde{D}'$ belong to the upper half-circles of $circ(\tilde{D})$ and $circ(\tilde{D}')$, respectively. Therefore these arcs intersect each other at most once within $\texttt{Cells}(\mathcal{S})$. If they do not intersect each other within $\texttt{Cells}(i+1, j-1)$ (see Figure 3(b)), then there must be a dominance relationship between them with respect to $[i+1, j-1]$. Moreover, as shown in Figure 3(a), if they do intersect each other within $\texttt{Cells}(i+1, j-1)$, then the subarc of the left-right arc of $\tilde{D}$ to the right of the intersection point is contained in $\tilde{D}'$ (or vice versa). □

Observation 3. *Let $\tilde{D}$ be a strong assisting disk, and suppose w.l.o.g. that $o(\tilde{D})$ is above $r(\tilde{D})$. Then $\tilde{D}$ intersects its right bounding disk above the line l.*

Proof. Let D be the right bounding disk of $\tilde{D}$, and let a and b denote the two intersection points of D and $\tilde{D}$, where $a = r(\tilde{D})$. By symmetry considerations, segments $\overline{o(\tilde{D}), a}$ and $\overline{o(D), b}$ are parallel. Therefore b must be above $o(D)$ which is above l. □

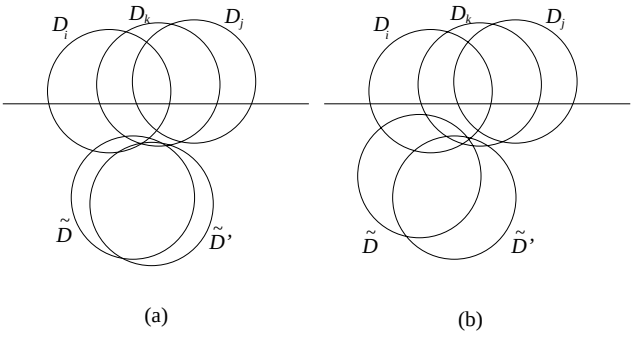


Fig. 3. Proof of Observation 2

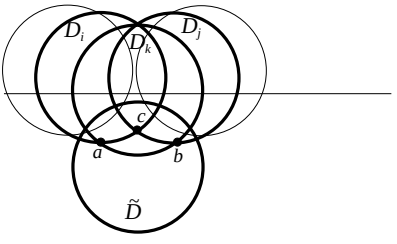


Fig. 4. $\{D_i, D_j, \tilde{D}\}$ is an assisting set for the interval $[i, j]$

Observation 4. *Consider an assisting disk $\tilde{D}$ that covers the intersection point of $arc(i)$ and $arc(j)$ of the chain $\mathcal{S}$, such that $left(\tilde{D}) \leq i$ and $right(\tilde{D}) \geq j$. Then $\{D_i, D_j, \tilde{D}\}$ is an assisting set for the interval $[i, j]$.*

Proof. Consider the disks D_i, D_j and D_k, where $i < k < j$. Consider an assisting disk $\tilde{D}$ that satisfies the conditions of Observation 4, i.e., $\tilde{D}$ contains the intersection point c of $arc(i)$ and $arc(j)$ and $left(\tilde{D}) \leq i$ and $right(\tilde{D}) \geq j$. Let a be the intersection point of D_i and D_k below l, and let b be the intersection point of D_k and D_j below l. As shown is Figure 4, $\tilde{D}$ intersects D_i in two points, with one point to the left of a and one point to the right of c. This implies that the arc between a and c is contained in $\tilde{D}$. Similarly, the arc between c and b is contained in $\tilde{D}$. Also $\tilde{D}$ intersects D_k in two points, with one point to the left of a and one point to the right of b. Therefore, all the area of $segment(k)$ outside $D_i \cup D_j$ is contained in $\tilde{D}$. □

We now show that the number of disks participating in the minimum assisted cover $\mathcal{F}$ is bounded by four times the size d^* of D^*, the minimum-cardinality cover of the points of $\mathcal{P}$ contained in B.

Lemma 5. *The set $\mathcal{U}'$ obtained by invoking Procedure* Single Line *satisfies $|\mathcal{U}'| \leq 2d^*$.*

Proof. Let $\mathcal{U}^*$ be the subset of $\mathcal{U}$ used in the optimal solution $\mathcal{D}^*$, and let $\mathcal{L}^*$ be the assisting disks used in $\mathcal{D}^*$. We now transform $\mathcal{U}^*$ into a set of boundary disks that, together with $\mathcal{L}^*$, forms an assisted cover.

If $\mathcal{U}^*$ contains non-boundary segments, then we replace each non-boundary segment s in $\mathcal{U}^*$ by the two boundary segments that contain it (see Observation 1). This manipulation results in a set $\mathcal{U}^{**}$ of boundary disks with $|\mathcal{U}^{**}| \leq 2|\mathcal{U}^*|$.

We now differentiate between the strong and weak assisting disks in $\mathcal{L}^*$. Set $\mathcal{U}^w = \emptyset$. Note that by Observation 2, the weak assisting disks $\tilde{D}$ in $\mathcal{L}^*$ can be ordered from left to right according to their leftmost intersection $l(\tilde{D})$ with the boundary of B. For each disk $\tilde{D}_i$ in this ordered set, if the left-right arcs of $\tilde{D}_i$ and $\tilde{D_{i+1}}$ intersect each other within B, then add a disk of $\mathcal{S}$ that contains this intersection point to the set $\mathcal{U}^w$. Otherwise, add $D_{right(\tilde{D}_i)}$ to $\mathcal{U}^w$.

For the strong assisting disks, let $\mathcal{U}^s$ be the set of their left and right bounding disks, i.e.,

$$\mathcal{U}^s = \{D_i \mid i = left(\tilde{D}) \text{ or } i = right(\tilde{D}) \text{ for some strong } \tilde{D} \in \mathcal{L}^*\} .$$

Consider the combined set of upper disks $\mathcal{U}^{**} \cup \mathcal{U}^w \cup \mathcal{U}^s$. We now show that this combined set forms together with $\mathcal{L}^*$ an assisted cover.

Consider two consecutive disks D_i and D_j in $\mathcal{U}^{**} \cup \mathcal{U}^w \cup \mathcal{U}^s$, under the ordering of the semi-chain $\mathcal{S}$. If there are occupied inner cells, i.e., if $\texttt{Occ}(i+1, j-1) \neq \emptyset$, then the points in these cells are covered in the optimal solution D^* by the disks of $\mathcal{L}^*$. We will show that a single disk of $\mathcal{L}^*$ is enough to cover the points in $\texttt{Occ}(i+1, j-1)$. Let $\mathcal{L}^*_{(i,j)}$ denote the set of disks in $\mathcal{L}^*$ that actually participate in covering the points in $\texttt{Occ}(i+1, j-1)$ in the optimal solution. (I.e., disks that are dominated with respect to $[i+1, j-1]$ are not taken into $\mathcal{L}^*_{(i,j)}$). If $\mathcal{L}^*_{(i,j)}$ includes a strong assisting disk $\tilde{D}$, we know that $left(\tilde{D})$ is outside the interval $[i+1, j-1]$ (because $D_{left(\tilde{D})} \in \mathcal{U}^s$ and $D_k \notin \mathcal{U}^s$ for $i < k < j$) and similarly $right(\tilde{D})$ is outside the interval $[i+1, j-1]$. Consider the following two cases. If both $left(\tilde{D})$ and $right(\tilde{D})$ are on the same side of $[i+1, j-1]$, then $\tilde{D}$ does not cover any internal points of $\texttt{Cells}(i+1, j-1)$ and thus cannot assist this interval. If $left(\tilde{D}) \leq i$ and $right(\tilde{D}) \geq j$, then by the definition of strong assisting disk and by Observation 4, we have that $\{D_i, D_j, \tilde{D}\}$ is an assisting set for this interval. Therefore, if such a strong assisting disks exists then no more disks are needed.

Otherwise, we know that $\mathcal{L}^*_{(i,j)}$ consists only of weak assisting disks and let $\tilde{D}$ be the leftmost disk in $\mathcal{L}^*_{(i,j)}$. But if more than one weak assisting disk is needed to cover the points in $\texttt{Occ}(i+1, j-1)$, then by Observation 2, the successor of $\tilde{D}$ in the weak assisting ordering, intersects $\tilde{D}$ within $\texttt{Cells}(i+1, j-1)$. This cannot happen since D_i and D_j are consecutive. (Note that the left-right arcs of $\tilde{D}$ and its successor are not disjoint, otherwise we would have added the right bounding disk of $\tilde{D}$).

We have shown that for each pair of consecutive disks $D_i, D_j \in \mathcal{U}^{**} \cup \mathcal{U}^w \cup \mathcal{U}^s$ there exists at most one assisting disk $\tilde{D} \in \mathcal{L}^*$ such that $\{D_i, D_j, \tilde{D}\}$ is an

assisting set for the interval $[i, j]$. We can now claim that the set $\mathcal{L}^* \cup \mathcal{U}^{**} \cup \mathcal{U}^w \cup \mathcal{U}^s$ forms an assisted cover $\mathcal{F}'$ and the number of left assisting pairs in $\mathcal{F}'$ is at most $|\mathcal{U}^{**} \cup \mathcal{U}^w \cup \mathcal{U}^s| \leq 2(|\mathcal{U}^*| + |\mathcal{L}^*|)$.

As Procedure Single Line finds a minimum-cardinality assisted cover $\mathcal{F}$ for $\mathcal{S}$, recalling Lemma 3 we have that $|\mathcal{U}'| = |\mathcal{F}| \leq |\mathcal{F}'| \leq 2(|\mathcal{U}^*| + |\mathcal{L}^*|) = 2d^*$. □

Corollary 1. *The number of disks of $\mathcal{D}$ participating in $\mathcal{F}$ is at most $4d^*$.*

Proof. The number of disks participating in $\mathcal{F}$ is $|\mathcal{U}'| + |\mathcal{L}'| \leq 2|\mathcal{U}'| \leq 4d^*$. □

The following theorem summarizes the result of this section.

Theorem 1. *One can compute a 4-approximation for the Single Line Problem by invoking Procedure* Single Line.

3 A 38-Approximation Algorithm for DUDC

In this section we present an approximation algorithm for our main problem: Given a set $\mathcal{P}$ of points in the plane and a set $\mathcal{D}$ of unit disks, find a subset $\mathcal{D}' \subseteq \mathcal{D}$ of minimum cardinality, such that $\mathcal{P} \subseteq \cup_{D \in \mathcal{D}'} D$. We show that this algorithm computes a 38-approximation.

The algorithm first lays a regular grid over the input scene, such that the distance between two consecutive vertical lines (alternatively, horizontal lines) is 3/2. Let $\mathcal{V}$ (resp., $\mathcal{H}$) be the set of vertical lines (resp., horizontal lines) of the grid.

The algorithm consists of two stages. In the first stage, for each line $l \in \mathcal{V} \cup \mathcal{H}$ such that there exists a disk in $\mathcal{D}$ that is intersected by l, we apply the 4-approximation algorithm for the single line problem (presented in Section 2) twice; once for each side of l. For a more detailed description, assume w.l.o.g. that l is vertical and let $\mathcal{D}_l^l \subseteq \mathcal{D}$ (resp., $\mathcal{D}_l^r \subseteq \mathcal{D}$) be the subset of disks that are intersected by l and whose centers lie to the left (resp., to the right) of l. We apply the 4-approximation algorithm twice. Once to the set $\mathcal{D}_l^l$ (using the disks in $\mathcal{D} \setminus \mathcal{D}_l^l$ as assisting disks), in order to cover the points in $\mathcal{P}$ that lie in the union of the disks in $\mathcal{D}_l^l$ and to the right of l, and once to the set $\mathcal{D}_l^r$ in order to cover the points in $\mathcal{P}$ that lie in the union of the disks in $\mathcal{D}_l^r$ and to the left of l.

Consider now an arbitrary point $p \in \mathcal{P}$. If there exists a disk in $\mathcal{D}$ that contains p and whose center does not lie in the same grid-square as p, then p is already covered in the first stage of the algorithm. Let $\mathcal{Q} \subseteq \mathcal{P}$ be the subset of points that are not yet covered. Then, for each $p \in \mathcal{Q}$, p can only be contained in disks whose centers lie in the grid-square of p. Thus, in the second stage, we consider each (non-empty) grid-square separately. For each such square $\mathcal{S}$ (of side length 3/2), we would like to cover the points in $\mathcal{Q} \cap S$ by disks whose centers lie in $\mathcal{S}$. This can be done by applying the 6-approximation algorithm described in Section 4.

3.1 Analysis

It is clear that at the end of the second stage each point in $\mathcal{P}$ is covered. We now prove that the size of the subset (i.e., cover) computed by our algorithm is

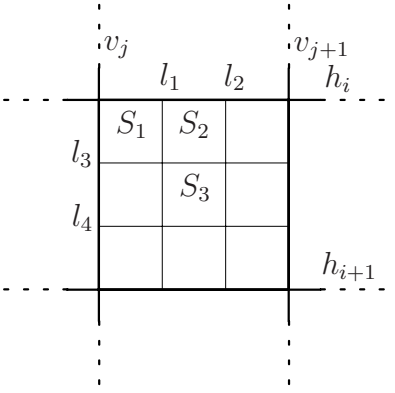


Fig. 5. The cell S

at most 38 times the size of an optimal cover. We first claim that a disk in $\mathcal{D}$ can participate in at most 8 applications of the algorithm of Section 2.

Claim. Let D be a disk in $\mathcal{D}$. Then, in the first stage of the above algorithm, D can participate in at most 8 applications of the algorithm of Section 2.

Proof. Let o be the center of D, and let S be the grid-square in which o lies. Divide S into 9 equal squares by lines l_1, l_2, l_3, and l_4, as depicted in Figure 5. We distinguish between three cases, depending on the location of o within S.

Case 1: $o \in S_1$ (or in any other corner sub-square of S). In this case D can participate as a non-assisting disk in at most 2 applications of the algorithm of Section 2 (namely, v_j (left) and h_i (up)). It can also participate as an assisting disk in at most 6 applications (namely, v_{j-1} (right), v_j (right), v_{j+1} (left), and h_{i-1} (down), h_i (down), h_{i+1} (up)). Thus in total D can participate in at most 8 applications of the algorithm of Section 2.
Case 2: $o \in S_3$ (i.e., in the middle sub-square of S). In this case D can participate as a non-assisting disk in 4 applications of the algorithm of Section 2 (namely, v_j (left), v_{j+1} (right), and h_i (up), h_{i+1} (down)). It can also participate as an assistant disk in 4 applications (namely, v_j (right), v_{j+1} (left), h_i (down), and h_{i+1} (up)). Thus in total D can participate in at most 8 applications.
Case 3: $o \in S_2$ (or in any other of the remaining sub-squares of S). In this case D can participate as a non-assisting disk in 3 applications of the algorithm of Section 2 (namely, v_j (left), v_{j+1} (right), and h_i (up)). It can also participate as an assisting disk in 5 applications of the algorithm of Section 2 (namely, v_j (right), v_{j+1} (left), and h_{i-1} (down), h_i (down), h_{i+1} (up)). Thus in total D can participate in at most 8 applications. □

Theorem 2. *The algorithm above computes a 38-approximation for* DUDC.

Proof. Consider a disk D in an optimal solution. By Claim 3.1 we know that D can contribute to the solution of at most eight single line problems and one

single square problem. Since each of these problems is solved separately, and since the approximation ratio for the single line problem is 4 and for the single square problem is 6, we obtain that the approximation ratio of the algorithm above is $8 \times 4 + 1 \times 6 = 38$. □

Remark. The choice of grid-square size $3/2 \times 3/2$ seems to be optimal (in our approach). By increasing the grid-square size one can reduce the number of applications of the single line algorithm a disk participates in. For example, for a square size 2×2 this number is 6, and for a square of size 3 this number is only 4. However, the approximation ratio for the single square problem increases, and the final approximation ratio that is obtained is greater than 38. Trying to decrease the squares to, e.g., $\sqrt{2} \times \sqrt{2}$ increases the number of applications of the single line algorithm a disk participates in to 10, which already gives a final approximation ratio that is greater than 38.

4 A 6-Approximation for the Single $\left(\frac{3}{2} \times \frac{3}{2}\right)$-Square Problem

Let S be a grid square of side length 3/2. In this section we devise a constant-factor approximation algorithm for covering the points $\mathcal{P}'$ of $\mathcal{P}$ that lie in S using the centers $\mathcal{O}'$ of $\mathcal{O}$ that lie in S; moreover we show that this constant is 6.

We divide the square S into nine equal squares by lines l_1, l_2, h_1 and h_2 as shown in Figure 6, and distinguish between the different cases according to the location of the center points of $\mathcal{O}'$ with respect to the nine subsquares. For each such case, we first check if there exists an optimal solution consisting of at most two centers, and if yes we return this solution. Otherwise, we apply the appropriate combination of claims from the following series of nine claims, and verify that by doing so we obtain an at most 6-approximation. For lack of space only the first two claims are included in this version.

Claim 1. *Any two centers o_i and o_j such that $o_i \in S_1$ and $o_j \in S_3$ satisfy $S_2 \subseteq D(o_i) \cup D(o_j)$, where $D(o)$ is a unit disk centered at o.*

Proof. Let p be a point in S_2, and assume w.l.o.g. that p lies in the right side of S_2. Then the disk $D(o_j)$ covers p. □

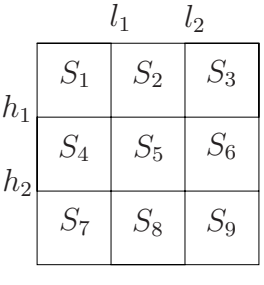


Fig. 6. Square S

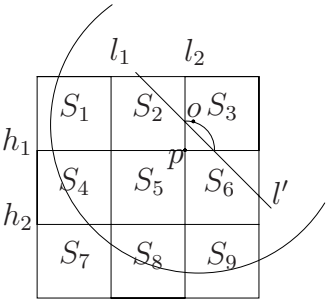


Fig. 7. Claim 2

Remark. Claim 1, as well as all subsequent claims, has several symmetric formulations.

Claim 2. *Consider the centers that lie in S_3 (alternatively S_1, S_7, or S_9). Then there exists a line l' such that all the centers in S_3 lie above l', and there exists a center o in S_3 such that all points in S_2 and S_6 above l' are covered by $D(o)$.*

Proof. Let o be the center in S_3 closest to the intersection point p of lines h_1 and l_2. Draw a disk centered at p of radius $d(p, o)$ (where $d(p, o)$ is the distance between p and o), and let l' be the line defined by the intersection points of this disk with the boundary of S_3. See Figure 7.

W.l.o.g. we show that $D(o)$ covers all points of S_6 above l'. Notice that the greatest distance between a point in the triangle formed in S_6 and a point on the arc is S_3 is determined by the intersection point of l' and the right side of S and the intersection point of l' and l_2; moreover this distance is actually the length of the diagonal of the squares S_i $(1 \leq i \leq 9)$ which is smaller than one. Therefore, regardless of the location of o on the arc, $D(o)$ covers all points of S_6 above l'. □

We now use the claims to obtain a 6-approximation for the singe $(3/2 \times 3/2)$-square problem. There are many cases to consider, depending on the location of the center points. We have generated all of the cases systematically, and have verified for each of them that an at most 6-approximation can be computed by applying the appropriate combination of claims from the series of claims. Due to the large number of cases and the resemblance between them, let us consider two cases for example.

Example 1. All the centers are in one square S_i $(1 \leq i \leq 9)$. In this case we optimally solve the problem of covering the points of $\mathcal{P}'$ using the centers in S_i, applying the algorithm of Lev-Tov [7] (that is not restricted to congruent disks).
Example 2. Assume all the centers are in squares S_1 and S_8. Apply Claim 2 to the square S_1 to obtain a line l' and a center $o \in S_1$, such that all the centers in S_1 are above l' and all points above l' in S_2 and in S_4 are covered by $D(o)$. We now apply the algorithm of Section 2 to the line l' using the centers in S_8 as

assisting centers. Finally, the remaining uncovered points are covered optimally using only the centers in S_8. It is easy to verify that the approximation factor in this case is 6 (actually, $5\frac{1}{3}$).

The following theorem summarizes the main result of this section.

Theorem 3. *One can compute a 6-approximation for the single* $(3/2 \times 3/2)$*-square problem.*

References

1. Brönnimann, H., Goodrich, M.T.: Almost optimal set covers in finite VC-dimension. Discrete Computational Geometry 14, 463–479 (1995)
2. Calinescu, G., Mandoiu, I.I., Wan, P.-J., Zelikovsky, A.: Selecting forwarding neighbors in wireless ad hoc networks. MONET 9(2), 101–111 (2004)
3. Clarkson, K.L., Varadarajan, K.: Improved approximation algorithms for geometric set cover. In: Proc. 21st ACM Sympos. Computational Geometry, pp. 135–141 (2005)
4. Gonzalez, T.: Covering a set of points in multidimensional space. Information Processing Letters 40, 181–188 (1991)
5. Hochbaum, D.S., Maas, W.: Approximation schemes for covering and packing problems in image processing and VLSI. J. ACM 32, 130–136 (1985)
6. Johnson, D.S.: The NP-completeness column: An ongoing guide. J. Algorithms 3, 182–195 (1982)
7. Lev-Tov, N.: Algorithms for Geometric Optimization Problems in Wireless Networks. Ph.D. Dissertation, Weizmann Institute of Science (2005)
8. Narayanappa, S., Vojtechovsky, P.: An improved approximation factor for the unit disk covering problem. In: Proc. 18th Canadian Conf. Computational Geometry, pp. 15–18 (2006)
9. Raz, R., Safra, S.: A sub-constant error-probability low-degree test, and a sub-constant error-probability PCP characterization of NP. In: Proc. 29th ACM Sympos. Theory of Computing, pp. 475–484 (1997)

Geodesic Disks and Clustering in a Simple Polygon*

Magdalene G. Borgelt, Marc van Kreveld, and Jun Luo

Dept. of Computer Science, Utrecht University
{magdalene,marc,ljroger}@cs.uu.nl

Abstract. Let P be a simple polygon of n vertices and let S be a set of N points lying in the interior of P. A *geodesic disk* $GD(p, r)$ with center p and radius r is the set of points in P that have a geodesic distance $\leq r$ from p (where the geodesic distance is the length of the shortest polygonal path connection that lies in P). In this paper we present an output sensitive algorithm for finding all N geodesic disks centered at the points of S, for a given value of r. Our algorithm runs in $O((n + (kn)^{\frac{2}{3}} + k) \log^c n)$ time, for some constant c and output size k. It is the basis of a cluster reporting algorithm where geodesic distances are used.

1 Introduction

Motivation. Clustering is the determination of relatively large subsets of a set that are in each other's proximity [12,13,14]. It is one of the most important and generally applicable techniques in data analysis. Often, the original data is a set of objects with various attribute values which are used as coordinates in a two- or higher-dimensional space. A cluster is a subset of the objects with similar attribute values, and a clustering of the objects is a partitioning into subsets so that objects in the same subset are similar, whereas objects in different subsets are not similar. To determine similarity, or distance, between two objects, a distance measure is needed, for which any L_p-metric (like the Euclidean metric) can be used.

In geographic situations, an important type of clustering is for sets of points in the real world [16,18]. The coordinates of the points do not stand for attribute values but for a specific location. Depending on the origin of the points, the real world may also include other objects like obstacles that influence the distance between points, and therefore the clusters. Obstacles can be bodies of water that are not crossed by certain land animals, or large open areas that are not used by forest animals like squirrels.

Suppose that we are given a set S of N points in a geographic situation like an island, and we wish to find clusters. Clusters are large enough subsets of S that lie within a region of a maximum radius. The value m that represents

* This research has been partially funded by the Netherlands Organisation for Scientific Research (NWO) under FOCUS/BRICKS grant number 642.065.503.

T. Tokuyama (Ed.): ISAAC 2007, LNCS 4835, pp. 656–667, 2007.

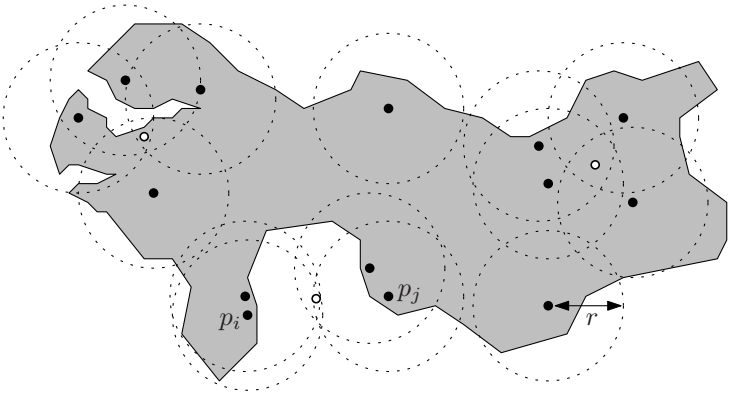


Fig. 1. Polygonal island with points, each with a radius-r Euclidean circle shown. The three open markers are centers of circles that contain four points. Only the rightmost one gives a cluster center if geodesic distances are used.

the minimum size of a cluster, and the radius r that represents the maximum extent of a cluster region, are fixed and specified by domain experts (biologists). Distances need to be measured by paths that go over the island only, which means that the length of a geodesic shortest path determines the distance. The geometric problem that arises is: Given a set S of N points inside a simple polygon P with n vertices, determine all subsets of S of size at least m for which a center point q exists that has geodesic distance at most r to all points in the subset (see Figure 1).

Related research. There is a large body of literature on clustering [19, 8, 20]. For more on clustering with respect to obstacles, see [6] for a recent survey. There are also several papers that compute clusters in a point set instead of a clustering of a point set [15, 5].

A recent approach to geographic clustering is given by Gudmundsson *et al.* [9]. The basic assumption is that in certain situations, there are regions where points can occur and regions where they cannot occur, and therefore a subset of points is a cluster if (i) the subset is large enough, (ii) the region has small enough radius, and (iii) the area where points can occur is small enough. The situation corresponds to nesting locations of sea birds on a group of islands, since nests cannot be in the water. Note that the Euclidean distance is still valid in this situation.

Problems concerning the geodesic distance with respect to a simple polygon have been studied mostly in the past. Among them are the computation of the geodesic center of a given simple polygon, and its geodesic diameter [4,17]. Given a simple polygon P with n edges in the plane and a set of point sites in its interior or on its perimeter, Aronov [3] studied computing the Voronoi diagram of the set of sites with respect to geodesic distance.

Results of this paper. We solve our cluster reporting problem using geodesic distances inside a polygon by inverting the problem: we generate boundaries of

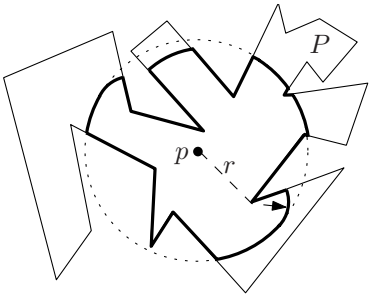


Fig. 2. Boundary of a geodesic disk in a simple polygon. The dotted circle is a normal radius-r circle.

radius-r geodesic disks centered at the points of S, and find points that lie inside at least m geodesic disks. A *geodesic disk* $GD(p, r)$ with radius r and center p is the set of points in P that have a geodesic distance $\leq r$ from p. It is easy to see that a geodesic disk is a shape that is bounded by circular arcs (not necessarily of the same radius) and pieces of the perimeter of P (see Figure 2). The main problem that arises is the computation of the N geodesic disks of the points of S inside polygon P. We present an *output-sensitive* algorithm for this problem that runs in $O((n + (kn)^{\frac{2}{3}} + k) \log^c n)$ time for some constant c and output size k. Note that $k = \Omega(N)$ and $O(n \cdot N)$. To appreciate this result, note that a direct approach to computing a geodesic disk would treat each point $p \in S$ separately by computing the shortest path tree of p inside P, and then determining the circular arcs and boundary parts inside each funnel [10]. This procedure takes $O(n)$ time per geodesic disk, and $O(n \cdot N)$ time in total. In our application we expect k to be significantly smaller than $\Theta(n \cdot N)$, hence the objective to design an output-sensitive algorithm.

Overview of this paper. The remainder of this paper is organized as follows. Section 2 gives the general algorithm to compute a geodesic disk in an output-sensitive manner. It is based on two data structures, which are presented in Section 3. In Section 4 we show how to produce the geodesic disk boundaries from the output of the queries. The analysis of the algorithm is given in Section 5, and Section 6 shows how the geodesic clusters are determined.

2 A Query Algorithm to Compute a Geodesic Disk Boundary

Let $S = \{p_1, p_2, \cdots, p_N\}$ be a set of N points inside or on the perimeter of a simple polygon $P = \{v_1, v_2, \cdots v_n\}$ whose n vertices are given in clockwise order. For a fixed real number r, we present an algorithm that computes all geodesic disks $GD(p_i, r)$, for $i = 1, \ldots, N$.

The boundary of P is denoted as ∂P. For two vertices $v_i, v_j \in \partial P$, let $\partial P[v_i, v_j]$ be the part of boundary of P in clockwise order from v_i to v_j. A ray $\overrightarrow{pv_i}$ is a

half line which starts at p and goes through v_i. For a point p in P that is visible from v_i, v_j, we have two rays $\overrightarrow{pv_i}$ and $\overrightarrow{pv_j}$. We use $wedge(p, v_i, v_j)$ to denote the wedge which starts from ray $\overrightarrow{pv_i}$ and rotates around p clockwise until it reaches ray $\overrightarrow{pv_j}$. Before illustrating our algorithm to compute a geodesic disk $GD(p, r)$ in P, we assume there exist two query algorithms that use preprocessed data structures. We will discuss the details of the data structures and query algorithms in Section 3.

1. $CLSF(p, v_i, v_j)$: the inputs are a point p in P and two vertices v_i, v_j of P, where p can see v_i, v_j. The query reports the closest line segment or vertex from p among the line segments and vertices of $\partial P[v_i, v_j]$ that are visible from p. If the output is a vertex, then we can use either line segment which is on $\partial P[v_i, v_j]$ and is incident to that vertex as the output.
2. $FVSP(p, v_i)$: the inputs are a point p in P and a vertex v_i of P, the query reports the first vertex of the shortest path from p to v_i.

The inputs of the algorithm $\boldsymbol{GD}(p, v_i, v_j, r')$ are a point p inside P, two vertices v_i, v_j of P such that p is visible to v_i, v_j, and the radius r'. The output is the set of line segments of $\partial P[v_i, v_j]$ such that the shortest path distances from p to those line segments are $\leq r'$. To compute the geodesic disk itself, some straightforward extra work is needed; this is deferred to the full paper. At the beginning, p is some point of S, $r' = r$, and $v_i = v_j$, where v_i is a vertex of P that is visible to p, which means the query range is the whole boundary of P. The algorithm runs as follows: using $CLSF(p, v_i, v_j)$ we find the closest line segment from p among all line segments of $\partial P[v_i, v_j]$. Let that closest line segment be $v_q v_{q+1}$. If the distance from p to $v_q v_{q+1}$ is larger than r, then we are done. Otherwise $v_q v_{q+1}$ is reported and there exists a closest point $a \in v_q v_{q+1}$.

Lemma 1. *The closest point a to p (by geodesic distance in P) of the line segment reported by $CLSF(p, v_i, v_j)$ is visible from p.*

Proof. Suppose the line segment $[v_q, v_{q+1}]$ is reported by $CLSF(p, v_i, v_j)$ and the closest point of $[v_q, v_{q+1}]$ to p is a. If a is not visible from p, then the first vertex v_b of the shortest path from p to a is inside or on the boundary of $wedge(p, v_i, v_j)$ and $v_b \in \partial P[v_i, v_j]$. So there is at least one line segment $[v_b, v_{b+1}]$ or $[v_{b-1}, v_b]$ which is $\in \partial P[v_i, v_j]$ that is closer to p than $[v_q, v_{q+1}]$. □

We assume without loss of generality that a is vertically above p (see Figure 3). The shortest path from p to v_q is a convex chain and its vertices are vertices of P. Let this convex chain be $pv_{l_1}v_{l_2}v_{l_3}\ldots v_{l_k}v_q$. We know that $pv_{l_1}v_{l_2}v_{l_3}\ldots v_{l_k}v_q$ is on the left side of the line though p, a. Similarly, we have the shortest path from p to v_{q+1} which is a convex chain $pv_{h_1}v_{h_2}v_{h_3}\ldots v_{h_{k'}}v_{q+1}$ on the right side of the line through p, a.

$\partial P[v_i, v_j]$ is partitioned into several parts: $\partial P[v_i, v_{l_1}]$, $\partial P[v_{l_1}, v_{l_2}]$, $\partial P[v_{l_2}, v_{l_3}]$, $\ldots$, $\partial P[v_{l_k}, v_q]$, $\partial P[v_{q+1}, v_{h_{k'}}]$, $\ldots, \partial P[v_{h_2}, v_{h_1}], \partial P[v_{h_1}, v_j]$; see Figure 3. They give rise to subproblems that we solve sequentially and recursively. Since the subproblems on the left are the same as those on the right, we discuss the situation

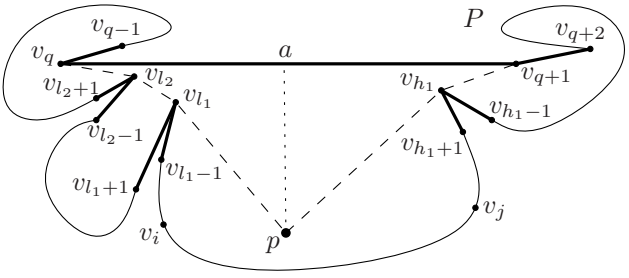


Fig. 3. Illustration of the algorithm

on the left. First, we solve subproblems $\boldsymbol{GD}(p, v_i, v_{l_1}, r)$ and $\boldsymbol{GD}(p, v_{h_1}, v_j, r)$ recursively. For all other parts $\partial P[v_{l_1}, v_{l_2}], \partial P[v_{l_2}, v_{l_3}], \ldots, \partial P[v_{l_k}, v_q]$, as long as the shortest path distance from p to v_{l_t} $(1 \leq t \leq k)$ is $< r$, we solve the subproblem $\boldsymbol{GD}(v_{l_t}, v_{l_t+1}, v_{l_{t+1}}, r - (|pv_{l_1}| + \cdots + |v_{l_{t-1}} v_{l_t}|))$ recursively. We don't need to compute the whole shortest path from p to v_q. We only need to find pv_{l_1} by $FVSP(p, v_q)$. If the distance from p to v_{l_1} is $< r$, then we find $v_{l_1} v_{l_2}$ by $FVSP(v_{l_1}, v_q)$, and so on.

There is one special case. If the output of $CLSF(p, v_i, v_j)$ is v_i or v_j, then there are two cases: (assume that v_i is the one reported by $CLSF(p, v_i, v_j)$)

1. If v_{i+1} is inside $wedge(p, v_i, v_j)$, then the algorithm continues normally.
2. If v_{i+1} is outside $wedge(p, v_i, v_j)$, then we need to do a ray shooting query with $\overrightarrow{pv_i}$. Suppose $\overrightarrow{pv_i}$ first hits $v_k v_{k+1}$, which is a line segment of $\partial P[v_i, v_j]$. Then the shortest paths from p to v_k and v_{k+1} separate $\partial P[v_i, v_j]$ into several subparts and the problem becomes several subproblems. We can solve those subproblems sequentially and recursively as above.

Lemma 2. *The algorithm $\boldsymbol{GD}(p, v_i, v_j, r')$ reports all line segments of $\partial P[v_i, v_j]$ that have geodesic distance at most r' from p at most once, and vertices at most twice.*

Proof. Omitted. □

3 Data Structures for *CLSF* and *FVSP*

In this section we describe the two data structures and query algorithms needed in the algorithm for geodesic disks. We also used a ray shooting data structure; this is standard with $O(\log n)$ query time in preprocessed simple polygons [7].

3.1 Closest Boundary Point in Subpolygon Queries

In this section we discuss how to find, for a given query point p and two vertices v_i and v_j of a simple polygon P, the point of the boundary of P between v_i and v_j that

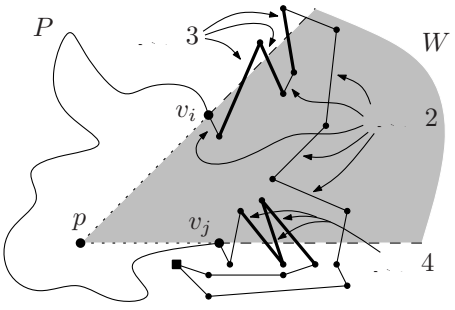


Fig. 4. Edges of P of cases 2, 3, and 4 (cases 3 and 4 are shown thicker). Note that the vertex of $\partial P[v_i, v_j]$ closest to p, the square, is not in W.

is closest to p. Vertices v_i and v_j can also be the answer to the query. We assume that pv_i and pv_j lie completely inside P (in other words: p sees v_i and v_j). To compute geodesic disks, we only need to solve the query problem with this restriction. The closest point q that is found must also be such that segment pq lies inside P.

Assume that some point q on the boundary of P between v_i and v_j is the point closest to p. Because pq lies inside P and p sees v_i and v_j, the angle of $\overrightarrow{pq}$ must be between the angles of $\overrightarrow{pv_i}$ and $\overrightarrow{pv_j}$.

Without the restriction that pq must be inside P, we could have built a binary search tree T on $v_1, \ldots, v_n$, and construct a Voronoi diagram preprocessed for planar point location as associated structure with every internal node of T. A query would be answered by determining the search paths in T to v_i and v_j, and for all maximal subtrees strictly between these search paths, query the associated structure. But then a point may be found that does not see p inside the whole polygon P (see Figure 4).

The solution is to adapt the data structure so that we only query inside the wedge W bounded by the rays $\overrightarrow{pv_i}$ and $\overrightarrow{pv_j}$. We have to take care to treat edges that lie partially inside this wedge correctly. We use five different data structures to handle all cases. In each case the main tree T is a binary search tree on $v_1, \ldots, v_n$, and the final associated structure is a planar point location structure on some Voronoi diagram. The first few associated structures (levels between the main tree and the point location structure) allow us to select the vertices or edges to which the case applies [1,2]. The five cases are the following.

Case 1: vertices inside W.
Case 2: edges of which both endpoints lie inside W.
Case 3: edges that intersect the ray $\overrightarrow{pv_i}$, have one endpoint inside W, and whose angle with $\overrightarrow{pv_i}$ is less than $\pi/2$, measured inside the wedge and closer to p (see Figure 4).
Case 4: edges that intersect the ray $\overrightarrow{pv_j}$, have one endpoint inside W, and whose angle with $\overrightarrow{pv_j}$ is less than $\pi/2$, measured inside the wedge and closer to p (see Figure 4).

Case 5: edges that intersect both rays $\overrightarrow{pv_i}$ and $\overrightarrow{pv_j}$, and both angles are less than $\pi/2$.

For the first case we use a partition tree as the main tree. For the second case we use two levels of partition trees. We treat the third case in more detail, the fourth case is the same and the fifth case can be treated in the same manner.

For the third case, let T be the main tree with $v_1, \ldots, v_n$ in the leaves. An internal node μ corresponds to a subchain $v_s, \ldots, v_t$ of the boundary of P. Let $E_\mu = \{v_s v_{s+1}, \ldots, v_{t-1} v_t\}$ be the edges in this subchain. To be able to select all edges that have one endpoint in the wedge W we take one endpoint of each edge and use a partition tree as the second level structure. To select the edges that intersect $\overrightarrow{pv_i}$ among these, we store the other endpoints in a partition tree as well as the third level structure, and the points dual to the supporting lines of the edges as the fourth level structure. In the fifth level structure we select further on the angle condition. This can be done using a binary search tree on the orientations of the edges. The sixth and last level structure is the point location structure on the Voronoi diagram of the edges. The fourth and fifth cases use similar multi-level trees.

For any query wedge, we can use the levels of the tree to select the edges for which each of the cases apply, and query in the Voronoi diagram to find the closest one. Each of the five cases may give an answer, and we can simply take the closest one as the actual closest vertex or edge. All structures use storage $O(n \log^c n)$ and query time $O(\sqrt{n} \log^c n)$ for some constant c. Combinations of cutting trees and partition trees allow us to get faster query times at the expense of storage and preprocessing [1,2]. For any $n \leq m \leq n^2$, we can get storage and preprocessing time of $O(m \log^c n)$ and query time $O((n/\sqrt{m}) \log^c n)$.

3.2 First Vertex of the Shortest Path Queries

We partition P into $O(n)$ geodesic triangles in $O(n)$ time [7]. The three vertices of a geodesic triangle are vertices of P. The three edge chains are three shortest concave paths inside P. In [7] Chazelle *et al.* show that any line segment interior to P crosses at most $O(\log n)$ geodesic triangles. So the line segment pa in Figure 3 crosses at most $O(\log n)$ geodesic triangles. We observe two properties about the intersections of pa with those $O(\log n)$ geodesic triangle edge chains.

Lemma 3. *pa only intersects each geodesic triangle edge chain at most once.*

Proof. Omitted. □

Lemma 4. *pa intersects at most two edge chains of one geodesic triangle.*

Proof. Omitted. □

Suppose pa crosses a geodesic triangle bcd. According to Lemmas 3 and 4, pa intersects one (if p is inside bcd) or two edge chains of bcd and pa only intersects

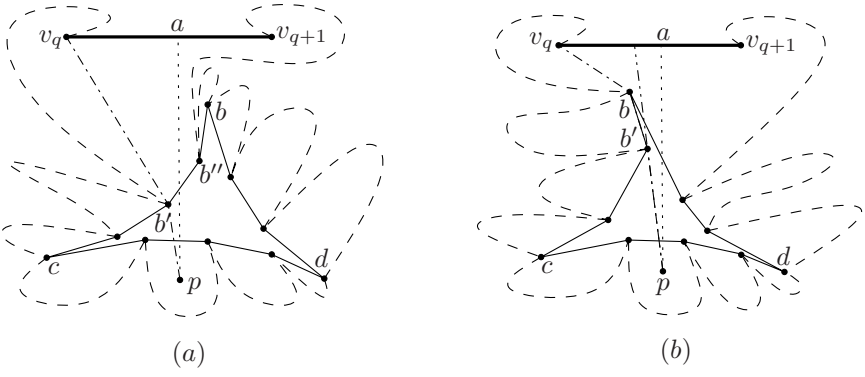


Fig. 5. (a) pa intersects the edge chain b–c. b' is the candidate for $FVSP(p, v_q)$. (b) pa does not intersect the edge chain b–c. The tangent point b' is the candidate for $FVSP(p, v_q)$.

those intersected edge chains once. For an intersected edge chain, suppose the intersected line segment is xx' and x is on the left side of pa and x' is on the right side of pa. Then x is a candidate for the first vertex of the shortest path from p to v_q, and x' is a candidate for first vertex of the shortest path from p to v_{q+1}. For a non-intersected edge chain, the vertex on a tangent line through p to this edge chain is also a candidate for the first vertex of the shortest path from p to v_q, provided the edge chain is on the left side of pa, and symmetrically, a non-intersected edge chain right of pa may provide a candidate for the first vertex of the shortest path from p to v_{q+1}. We explain those two cases by focusing on one edge chain b–c:

1. pa intersects the edge chain b–c, see Figure 5(a). Suppose the line segment of the edge chain from b to c intersecting pa is $b'b''$, and b' is on the same side of the line through p and a as v_q. Then b' is the only possible vertex from b–c that can be the first vertex of the shortest path from p to v_q. Given the edge chain b–c, we can find b' in $O(\log n)$ time.
2. pa does not intersect the edge chain b–c and b–c is on the same side of the line through p and a as v_q, see Figure 5(b). The only candidate for the first vertex from p to v_q on the edge chain b–c is the vertex on a line through p that is tangent to the edge chain b–c. Given the edge chain b–c, we can find that tangent vertex in $O(\log n)$ time.

Any geodesic triangle intersecting pa gives at most three candidate vertices. To decide whether a candidate vertex y is the first vertex of the shortest path from p to v_q, we do a ray shooting query with $\overrightarrow{py}$. The first vertex of the shortest path from p to v_q is y if and only if py does not intersect any other line segment of ∂P and the first intersection point of $\overrightarrow{py}$ after y with ∂P is on $v_q v_{q+1}$. This can also be tested in $O(\log n)$ time. Since there are $O(\log n)$ geodesic triangles we need to check, the total running time of $FVSP(p, v_q)$ is $O(\log^2 n)$.

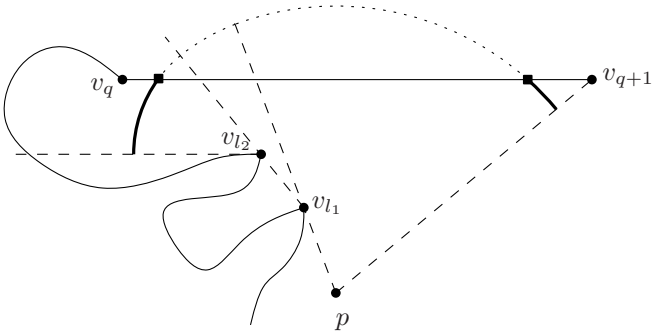


Fig. 6. Finding the intersection points of the edges of P with the circular arcs that form the geodesic disk boundary

4 Computing the Geodesic Disk Boundary

The algorithm as presented so far finds the set of edges of P that have some point at geodesic distance at most r from p, and a set of circular arcs that contain points at geodesic distance exactly r from p. To determine the boundary of the geodesic disk itself, we must combine this information into a simple polygon that has straight edges and circular arcs. Whenever an edge $v_q v_{q+1}$ is detected to be part of the boundary of the geodesic disk, we have the point p (or a vertex of P) which is the query center, and we have a wedge W in which the circular arc of radius r is valid. The following cases can be distinguished (we only treat the case of the left of the closest point a, like in Section 2):

> If $v_q v_{q+1}$ intersects the circular arc inside W, then we have found the (left) intersection point on $v_q v_{q+1}$ that gives a vertex of the geodesic disk.
> If $v_q v_{q+1}$ does not intersect the circular arc inside W, but W contains all of $v_q v_{q+1}$ left of point a, then all of $v_q a$ is part of the boundary of the geodesic disk.
> Otherwise, we repeat the above tests iteratively with v_{l_1}, v_{l_2}, ..., until we either find an intersection point left of a, or discover that all of $v_q a$ is part of the boundary of the geodesic disk.

In Figure 6, the edge $v_q v_{q+1}$ is found when p was the query center. The dashed lines show the wedge in which p is valid as an arc center, and we test whether the edge $v_q v_{q+1}$ intersects the circular arc inside the wedge. In the figure we only find the intersection point (square) when we test with v_{l_2}.

The tests can easily be integrated into the algorithm that finds the edges of P within geodesic distance r. Hence, the algorithm can also determine the boundary of the geodesic disk centered at p.

5 Complexity Analysis

If the output size of N geodesic disks is $O(k)$, then the algorithm will perform $O(k)$ *FVSP* and *CLSF* queries. The preprocessing is $O(n)$ time, plus the

time needed to build the multi-level trees of Subsection 3.1. We observed that a preprocessing time/query time trade-off exists: $O(m \log^c n)$ preprocessing time leads to $O((n/\sqrt{m}) \log^c n)$ query time. Assume we know k in advance. Then we can choose m to be such that the total query time and preprocessing time are of the same order: $k \cdot (n/\sqrt{m}) \log^c n = m \log^c n$, giving $m = (kn)^{\frac{2}{3}}$ (provided $n \leq m \leq n^2$).

Unfortunately, the output size k is not known, so we can not balance query time and preprocessing time easily. To overcome this problem, we will guess k, run the algorithm, and if it turns out that the guess was too low, we double our guess of k and start again: We build a data structure with slightly higher preprocessing time and slightly faster queries. Our initial guess is $k' = \max(n^{\frac{1}{3}}, 2N)$, since we know that $k \geq N$. Since we also know that $k \leq n \cdot N$, we will restart the algorithm at most $\log_2 n$ times. The running time is $O((n + (k'n)^{\frac{2}{3}} + k') \log^c n)$ in each round. Summation over the rounds yields $O((n + (kn)^{\frac{2}{3}} + k) \log^c n)$ time, for some constant c.

The adaptations made to find the boundaries of the geodesic disks themselves do not influence the asymptotic running time.

Theorem 1. *Given a simple polygon P with n vertices, a set S of N points inside P, and a positive real r, all N geodesic disks centered at the points of S can be computed in $O((n + (kn)^{\frac{2}{3}} + k) \log^c n)$ time, for some constant c, where k is the total boundary complexity of the geodesic disks.*

6 Geodesic Clustering in a Simple Polygon

In this section, we will show how to solve the geodesic clustering problem: given a simple polygon P with n edges, a set S of N points inside P, a radius r, and a subset size m, find all geodesic disks with radius r which contain at least m points of S. We define a *geodesic cluster center* as a point p in P such that $GD(p, r)$ contains at least m points of S. We define two cluster centers to be *distinct* if they contain different subsets of S, otherwise they are *equivalent*.

We compute N geodesic disks with the algorithm described before. Suppose the complexity of the N geodesic disks is $O(k)$; recall that $k = \Omega(N)$ and $k = O(n \cdot N)$. We can compute the arrangement of the geodesic disks in $O(k \log k + K)$ time and $O(k + K)$ space, where K is the number of intersection points in the arrangement [11]. Since any two geodesic disk boundaries can have at most two proper intersections, $K = O(k + N^2)$. One can expect that for the cluster reporting application, K is considerably smaller than quadratic in N.

After building the arrangement, we determine for each cell by how many geodesic disks it is covered. For one cell we do this brute-force in $O(k)$ time. After that, we do an arrangement traversal (e.g., depth-first) to visit all cells. If we cross a cell boundary that adds a geodesic disk to the cover, we put a one higher value in the cell, and otherwise we put a one lower value in the cell. Hence, entering an adjacent cell and determining the value takes only $O(1)$ time. Therefore, the whole traversal takes time linear in the number of cells.

Reporting all distinct cluster centers comes down to identifying the cells with value at least m. All points inside such a cell are equivalent cluster centers.

Theorem 2. *Given a simple polygon P with n vertices, a set S of N points inside P, a positive real r, and a positive integer m, all distinct cluster centers can be reported in $O((n+(kn)^{\frac{2}{3}}+k)\log^c n+k\log k+K)$ time, for some constant c, where k is the total boundary complexity of the geodesic disks, and K is the number of intersection points in the arrangement of N geodesic disks.*

7 Conclusions

The main contribution of this paper is a new algorithm to determine N geodesic disks inside a simple polygon with n vertices. Instead of treating every point separately, we use preprocessed data structures, which made it possible to develop an output-sensitive algorithm for computing the geodesic disks. If the output size is k, then the running time is $O((n+(kn)^{\frac{2}{3}}+k)\log^c n)$ time, for some constant c. We used this algorithm to solve a cluster reporting problem: find all subsets of the points of at least some size that lie inside a geodesic disk with radius at most a specified value.

Although the geodesic disk algorithm is output-sensitive, the cluster reporting algorithm is not in the true sense. It is an open problem to determine clusters with a truely output-sensitive algorithm. Also, the output-sensitive algorithm for geodesic disks does not have the desired running time of the form $O(f(n)+k)$ or $O(f(n)+k\log n)$, where k is the output size and $f(n)=o(n\cdot N)$. It is also an open problem to find such an algorithm. An important extension of our research is to deal with polygons that have holes, or, a set of polygonal obstacles. It is unclear how to design any output-sensitive algorithm for this case.

References

1. Agarwal, P.K.: Range searching. In: Goodman, J.E., O'Rourke, J. (eds.) Handbook of Discrete and Computational Geometry, ch. 36, 2nd edn., pp. 809–838. Chapman & Hall/CRC, Boca Raton (2004)
2. Agarwal, P.K., Erickson, J.: Geometric range searching and its relatives. In: Chazelle, B., Goodman, J.E., Pollack, R. (eds.) Advances in Discrete and Computational Geometry, AMS, Providence, RI. Contemporary Mathematics, vol. 223, pp. 1–56 (1999)
3. Aronov, B.: On the Geodesic Voronoi Diagram of Point Sites in a Simple Polygon. Algorithmica 4, 109–140 (1989)
4. Asano, T., Toussaint, G.: Computing the Geodesic Center of a Simple Polygon. In: Johnson, D.S., Nozaki, A., Nishizeki, T., Willis, H. (eds.) Perspectives in Computing: Discrete Algorithms and Complexity, pp. 65–79 (1987)
5. Aurenhammer, F.: Voronoi diagrams: A survey of a fundamental geometric data structure. ACM Comput. Surveys 23, 345–405 (1991)
6. Berkhin, P.: A Survey of Clustering Data Mining Techniques. In: Jacob, K., Charles, N., Marc, T. (eds.) Grouping Multidimensional Data: Recent Advances in Clustering, pp. 25–71 (2006)

7. Chazelle, B., Edelsbrunner, H., Grigni, M., Guibas, L., Hershberger, J., Sharir, M., Snoeyink, J.: Ray Shooting in Polygons Using Geodesic Triangulations. Algorithmica 12, 54–68 (1994)
8. Estivill-Castro, V., Lee, I.: Autoclust+: Automatic clustering of point-data sets in the presence of obstacles. In: TSDM 2000: Proc. of the First Int. Workshop on Temporal, Spatial, and Spatio-Temporal Data Mining-Revised Papers, pp. 133–146 (2001)
9. Gudmundsson, J., van Kreveld, M., Narasimhan, G.: Region-restricted clustering for geographic data mining. In: Proc. 14th Europ. Sympos. Algorithms, pp. 399–410 (2006)
10. Guibas, L., Hershberger, J., Leven, D., Sharir, M., Tarjan, R.: Linear-time Algorithms for Visibility and Shortest Path Problems inside Triangulated Simple Polygons. Algorithmica 2, 209–233 (1987)
11. Halperin, D.: Arrangements. In: Goodman, J.E., ORourke, J. (eds.) Handbook of Discrete and Computational Geometry, ch. 24, 2nd edn., pp. 529–562. Chapman & Hall/CRC, Boca Raton (2004)
12. Han, J., Kamber, M.: Data Mining: Concepts and Techniques. Academic Press, London (2001)
13. Hartigan, J.: Clustering Algorithms. John Wiley & Sons, New York (1975)
14. Jain, A., Dubes, R.: Algorithms for Clustering Data. Prentice-Hall, Englewood Cliffs (1988)
15. Laube, P., van Kreveld, M., Imfeld, S.: Finding REMO – detecting relative motion patterns in geospatial lifelines. In: Fisher, P.F. (ed.) Developments in Spatial Data Handling: proc. 11th Int. Sympos., pp. 201–215 (2004)
16. Miller, H.J., Han, J.: Geographic Data Mining and Knowledge Discovery. Taylor & Francis, London (2001)
17. Pollack, R., Sharir, M., Rote, G.: Computing the Geodesic Center of a Simple Polygon. Discr. & Comput. Geometry 4, 611–626 (1989)
18. O'Sullivan, D., Unwin, D.: Geographic Information Analysis. John Wiley & Sons, Hoboken, NJ (2003)
19. Tung, A.K.H., Hou, J., Han, J.: Spatial clustering in the presence of obstacles. In: Proc. of the 17th Int. Conference on Data Engineering, pp. 359–367 (2001)
20. Zaïane, O.R., Lee, C.: Clustering spatial data in the presence of obstacles: a density-based approach. In: Proc. of the 2002 Int. Symposium on Database Engineering & Applications, pp. 214–223 (2002)

An $O(n^2 \log n)$ Time Algorithm for Computing Shortest Paths Amidst Growing Discs in the Plane

Anil Maheshwari, Doron Nussbaum, Jörg-Rüdiger Sack, and Jiehua Yi

Carleton University, Ottawa ON, Canada

Abstract. We present an algorithm to compute a shortest path for a robot between two points that avoids n discs growing at a common speed in the plane. Our algorithm runs in $O(n^2 \log n)$ time, thus improving upon the best previous solution by a factor of n. The complexity for the growing disc problem matches the known bound for the more restricted case when the discs are static.

1 Introduction

A well-studied and fundamental problem in computational geometry is the determination of shortest paths (see e.g., [5] for a survey). It finds applications in a number of key areas such as GIS, VLSI design, graphics, computer games, and robotics. In robotics, the problem comes up in the context of planning collision-free motions for a robot operating in an environment containing obstacles. An important algorithmic challenge arises when the obstacles themselves move, as is the case in dynamic environments. This challenge is increased further, if the obstacle motion is unpredictable.

A recently introduced **model** [6,7], presents a first step to capture such unpredictability. The unpredictability of the obstacle locations is modeled by discs growing over time at some (maximum) speed. The velocity by which the discs grow is assumed to be a constant and to be less than the maximum velocity of the robot. Solutions are typically given in configuration space, where a circular robot shrinks to a point and obstacles are augmented appropriately by the radius of the robot. One is therefore interested in finding shortest paths which are guaranteed to be collision-free for a point robot operating in an unpredictable environment.

We state the resulting geometric problem, called **shortest path among growing discs** to be addressed in this paper: given a point robot, moving with maximum velocity V and a set of discs $\mathscr{D} = \{C_0, C_1, ..., C_n\}$ in Euclidean space, where the radii of the discs grow at speed v, with $v < V$, find a shortest path from a start location s to goal location g for the robot that avoids the growing discs. This problem has been introduced by van den Berg and Overmars [6,7]. Their $O(n^3 \log n)$ solution is employed in the context of motion planning in virtual environments such as commercial video games.

T. Tokuyama (Ed.): ISAAC 2007, LNCS 4835, pp. 668–680, 2007.

In this paper, we present an $O(n^2 \log n)$ algorithm for the shortest path among growing discs problem. By deriving some geometric insights into shortest paths among growing discs and, as tools, using a circular sweep technique and additively weighted Voronoi diagrams, we are able to reduce the existing $O(n^3 \log n)$ solution to $O(n^2 \log n)$. In [4], an $O(n^2 \log n)$ algorithm to find a shortest path from s to g among a set of discs is presented where the discs are static. Our solution for the growing disc problem instance therefore matches the complexity derived for the simpler static (fixed disc) case [2].

Section 2 presents preliminaries, Section 3 outlines our algorithms, Section 4.1 presents the preprocessing step, and the key algorithms are described in the remainder of Section 4. We conclude with Section 5

2 Preliminaries

In this section we introduce notion beginning with a **formal definition** for the shortest path among growing discs problem.

Let $\mathscr{D} = \{C_0, C_1, ..., C_n\}$ be a set of discs. Disc $C_i \in \mathscr{D}$ at time t is denoted by $C_i(t) = (O_i, r_i(t))$ with center O_i and a radius $r_i(t)$ at time t. A path π is a function $[t_a, t_b] \rightarrow \mathscr{R}^2$ where $\pi(t_x)$ denotes the location of the robot R, at time t_x. Thus, at the initial time, t_a, R is located at $\pi(t_a)$ and R reaches $\pi(t_b)$ at time t_b. A point $\pi(t_x)$ is collision-free with respect to $\mathscr{D}$ at time t_x if $\pi(t_x)$ is outside $\bigcup C_i(t_x), i = 0, ..., n$. A path π from $\pi(t_a)$ to $\pi(t_b)$ is collision-free if every point $\pi(t_x)$ on π is collision-free at time t_x, where t_x in $[t_a, t_b]$. The problem of finding a **shortest collision-free path** from a start location s to a goal location g in $\mathscr{R}^2$ is to find a collision-free path π: $[t_s, t_g] \rightarrow \mathscr{R}^2$ which minimizes $|t_g - t_s|$.

Since the discs grow over time, it is easily seen that the robot should always move at maximal velocity. Thus, if we know a path together with its starting time, we can determine its arrival time at any point on the path. In particular, segments on the robot's path are time encapsulated, but, for ease of notation, the time parameter t is omitted when no ambiguity arises.

To solve the problem, van den Berg and Overmars [6,7] use a cone model with time as the third dimension and apices located at the centers of the discs. They show that any shortest path from s to g for the robot is composed of alternating straight line segments of slope $\frac{v}{V}$ which are tangent to pairs of cones and logarithmic spiral segments which lie on the surface of each individual cone.

We solve the problem in 2-d thereby avoiding 3-dimensional geometric objects, operations and primitives. We first focus on the tangents between growing disk. A tangent line segment l, also called *tangent*, between two discs $C_i(t_p)$ and $C_j(t_q)$ is a line segment from a tangent point p on C_i to a tangent point q on C_j, such that for robot R traveling on l, it holds $l(t_p)$=p and $l(t_q)$=q. The *direction* of l (from p to q) is written as $\overrightarrow{l}$.

A tangent half line l_H is a half-line with endpoint originating at p and directed along $\overrightarrow{l_H}$. A tangent $l(= \overline{pq})$ from $p \in C_i$ to $q \in C_j$ that is collision free with respect to other growing discs is called a *valid tangent*. There are four types of tangents between a pair of discs C_i and C_j with starting time T_0 at disc

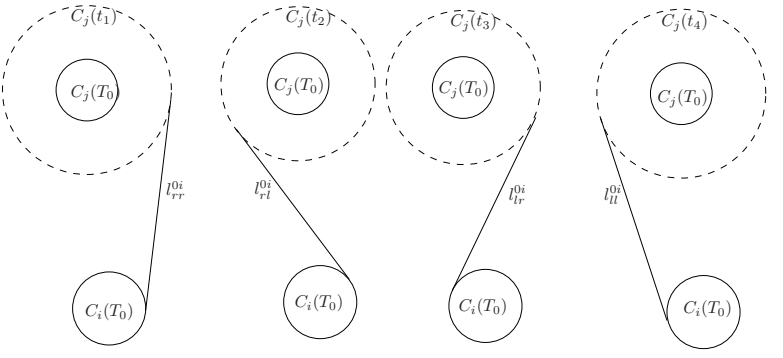


Fig. 1. Right-right, right-left, left-right, left-left tangents from disc $C_i(T_0)$ to disc C_j, which is originally $C_j(T_0)$

C_i. These are: right-right, right-left, left-right, and left-left tangents, denoted by $l_{rr}^{ij}(t)$, $l_{rl}^{ij}(t)$, $l_{lr}^{ij}(t)$, and $l_{ll}^{ij}(t)$, respectively, $t \geq T_0$. Knowing that these are time-encapsulated, we often use the short forms l_{rr}^{ij}, l_{rl}^{ij}, l_{lr}^{ij}, and l_{ll}^{ij}.

Refer to Figure 1. The four tangents are divided into two groups: one group, discussed here, consists of the right-right and right-left tangents, and the other group, which can be handled analogously, consists of the left-left and left-right tangents.

As functions of time, the endpoints of the tangents on some starting growing disc C_i form continuous curves called *departure curves* and the corresponding endpoints on the growing arrival disc C_j form *arrival curves*. A departure curve for a particular tangent is the trace, over time, of the tangent's endpoints that are located on the starting disc as the disc grows. Similarly, an arrival curve for the tangent is the trace of the tangent's endpoints on the growing arrival disc.

Departure curves for right-right and left-left tangents from C_i to C_j are straight lines. Those for right-left and left-right tangents are projections of 3-d sin-like curves [7].

Each departure curve is cut by at most $n-1$ growing discs which are defining $O(n)$ intervals. However, we will identify, for each disc pair, only one interval referred to as buffer. A buffer has the property that any shortest path passing through a departure curve must pass through it. For a pair of discs C_i and C_j, the *buffer (or buffer zone)* is the set of endpoints on the departure curve of C_i of all valid tangents for C_i, C_j, refer to Figure 2. It is easy to see that each buffer zone for a pair of growing discs forms an interval. Each interval corresponds to a unique time interval. We will use time or geometric description of the buffer zone as appropriate for the discussion.

Let the departure curve and the arrival curve for a pair of discs C_i and C_j be μ and ν, respectively. The buffer on the departure curve μ has two endpoints (if they exist), called the left endpoint, $\mu(T_0)$, and the right endpoint, $\mu(T_r)$, with two associated critical times, the starting time T_0 and T_r. There is a collision free tangent from a point on $\mu(t)$ to a point on ν for all times t, for which $T_0 \leq t \leq T_r$. T_r is denoted by $T_r(l_{rr}^{ij})$ ($T_r(l_{rl}^{ij})$) for l_{rr}^{ij} (l_{rl}^{ij}, respectively).

We call the disc C_0 which contains the starting points of tangents. The tangents originating from C_0 are sorted in counter clockwise (CCW) order by CCW angles. These angles are between the horizontal line passing through the center of C_0 (O_0) in direction $+\infty$ and each tangent. The tangent points on C_0 have the same order as the order of the tangents. Define *slope* as the angle of a tangent. Around C_0, starting from direction $+\infty$, in CCW order, the slopes of the tangents increase. The slope of a line l oriented from some point a on l to point b on l is defined as the CCW angle from the positive x-axis to $\overline{ab}$.

Let l_H be a tangent half line from C_0 to C_i. If the projection of O_j onto l_H lies on the corresponding tangent l, then C_j is *lower* than C_i along $\overrightarrow{l}$. Otherwise, C_j is *higher* than C_i along $\overrightarrow{l}$.

A point p is said to be to the left (right) of a segment $\overline{ab}$ if its projection is on $\overline{ab}$ and a, b, p is a left-turn (right-turn). A disc is to the left (right) of $\overline{ab}$ if its center is to the left (right) of $\overline{ab}$. (We will only consider discs that do not intersect $\overline{ab}$.) A line is to the left (right) of a segment $\overline{ab}$ if all its points are either to the left (right) of $\overline{ab}$ or their projections are outside $\overline{ab}$.

Let l_H be a tangent half line from C_0. The *distance* from $C_i(t)$ to l_H is defined as $|O_i p| - r_i(t_p)$, where, p is the projection of O_i on l_H, t_p is the time that the robot arrives at point p from the tangent point on C_0 at time t along $\overrightarrow{l_H}$.

Lemma 1. *Let l be an oriented line located to the left of l_{ll}^{ij}. If $l_{ll}^{ij} \parallel l$, then C_i and C_j are equi-distant to l. If the slope of l_{ll}^{ij} is smaller than that of l, then C_i is closer to l than C_j. Otherwise, C_j is closer.*

Lemma 2. *Let l_{rr}^{ij} be valid. Then, for any $T_0 < t \leq T_r$, $l_{rr}^{ij}(t) \parallel l_{rr}^{ij}(T_0)$, and $l_{rr}^{ij}(t)$ is located to the right of $l_{rr}^{ij}(T_0)$, where, T_0 and T_r are the two time values determining the buffer zone of l_{rr}^{ij}.*

It is well known that the Voronoi diagram of a set of discs does not change when the discs are growing at the same speed. Each departure curve μ on a disc C_i intersects the boundary of the Voronoi cell of C_i (written as V_{C_i}), at some point p, and the corresponding arrival curve ν on a disc C_j intersects the boundary of V_{C_j} at some point q. Points p and q define two tangents. One starts from $p \in \mu$ and arrives at a point on ν. The other one starts from $p' \in \mu$ and arrives at $q \in \nu$, refer to Figure 2. This determines two time values, say T_p and $T_{p'}$, associated with p and p', respectively.

Lemma 3. *Let l be a tangent from C_i to C_j. Let T_p be the time that the departure curve of l intersects with the boundary of V_{C_i}, and $T_{p'}$ be the starting time of l on C_i such that the intersection of l with the arrival curve on C_j is on the boundary of V_{C_j}. Then, for the maximum time value, $T_r(l)$, of the buffer zone of l holds: $T_r(l) \leq \min\{T_p, T_{p'}\}$.*

3 An Overview of Our Approach

Our approach is based on constructing a graph which, during run-time, will contain all shortest path information required to solve shortest path among growing discs

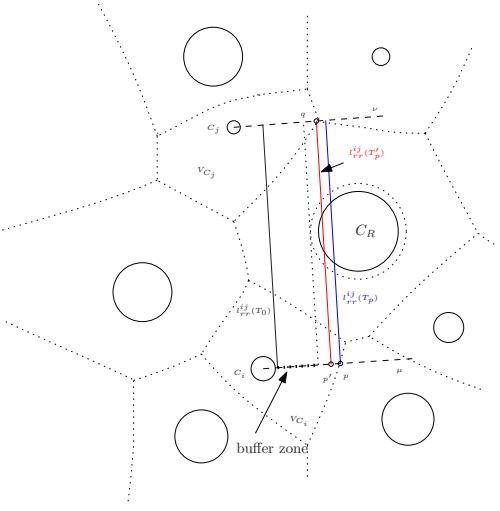


Fig. 2. The Voronoi diagram of a set of discs. The departure curve μ and arrival curve ν for C_i and C_j, and tangent $l_{rr}^{ij}(T_0)$. Tangent $l_{rr}^{ij}(T_p)$ intersects the Voronoi cell of C_i with p. Tangent $l_{rr}^{ij}(T'_p)$ intersects the Voronoi cell of C_j with point q. $p \in \mu$, $p' \in \mu$, $q \in \nu$. The bold segment on μ is the buffer zone for l_{rr}^{ij}, and C_R is the determining disc.

problem. In contrast to [6,7] though, our graph has substantially fewer vertices and edges. We review their graph construction first. In their cone-based approach, each departure curve is cut by at most $n-1$ cones thus defining $O(n)$ intervals. Each interval becomes a *node* in a graph G. For each interval, only the collision free path that arrives at the interval earliest is kept. Each node has two *edges* in G: a collision free tangent between two cones, and a piece of spiral segment between two neighbouring departure curves on a cone. The edges in G are obtained by computing the arrangement of the cones and the departure curves, and the trapezoidal map for the arrangement. There are $O(n^3)$ nodes and $O(n^3)$ edges in the graph G, thus applying Dijkstra's algorithm they obtain an $O(n^3 \log n)$ solution.

Our technique is based on the fact that in 3-d, along the Z direction, the projection of the intersection of the set of cones, whose bases are discs and parallel to the $X-Y$ plane, and apices located below the bases along the Z direction, is a Voronoi diagram in 2-d. The Voronoi diagram is the same as the Voronoi diagram of the set of discs obtained by cutting the cones with a horizontal plane. The Voronoi diagram plays a key role in our algorithm, thus avoiding the computation costs of the arrangement of the set of cones. For a departure curve, we compute its buffer zone (defined above) which has the property that any shortest path passing through a departure curve must pass through it. We efficiently pre-compute the buffer zones on each disc by deriving some geometric insights into shortest paths among growing discs. By employing buffers zones, we can decease the number of nodes and edges in the graph G to $O(n^2)$, and thus are able to reduce the existing $O(n^3 \log n)$ solution to $O(n^2 \log n)$. The main steps of our algorithm are:

Step 1: Construct a graph $G = (V, E)$ for shortest path computation

$G =:$ Nodes $V :=$ {all departure curves on discs, s and g}
Edges: $E := E_1 \bigcup E_2, where$
$E_1 :=$ {spirals between two neighbouring departure curves around each disc}
$E_2 :=$ {selected edges between discs}

Step 2: Efficient construction of edges in E_2

2-1. Compute the valid tangents between each pair of discs (see Section 4.1).

2-2. For all valid tangents compute the buffer zone (see the remainder of Section 4).

2-3. For every buffer zone define an edge (partial) in E_2 from the node corresponding to the departure curve associated with the buffer to the corresponding disc (at run time, the exact edge and the arriving node will be known). With each edge associate the buffer zone, i.e., a time interval. Edge costs are determined at run time.

Step 3: With each spiral edge associate its buffer zone, i.e., a time interval after which the edge expires. Edge costs are determined at run time.

Step 4: Find time-minimal shortest path from s to g in G.

Recall that there are four types of tangents between any two discs. Therefore, there are four types of departure curves and we will describe the computation of the buffer zones of the right-right and right-left departure curves (the left-right and left-left departure curves are handled analogously).

The buffer zone computation depends on the following fact: if $\bar{t}$ is the last starting time for a tangent l, between C_0 and C_i to be valid, i.e., l becomes invalid at any time $t' > \bar{t}$, then, there must be a third disc, say C_j, such that C_0, C_j and C_i are co-tangent. Thus, the key in the buffer zone computation is to find such C_j for l.

First, the departure curve for a pair of discs C_0 and C_i is trimmed by the Voronoi cells of C_0 and C_i, respectively, since, the starting point of a tangent on C_0 must be located inside the Voronoi cell of C_0, and the arriving point of the tangent on C_i must be inside the Voronoi cell of C_i. The arriving point has an associated starting time. The buffer zone is initially set to the minimum of the two starting times (see Lemma 3). The buffer zones of the valid right-right and right-left tangents are computed separately. For the valid right-left tangent l, which is tangent to C_0 and C_i, the buffer zone may be affected by a disc to the left of l or a disc to the right of l. The buffer zone is the minimum time among the three values: (i) the time restricted by the Voronoi cells, (ii) the time when C_0, disc to the left of l, and C_i are tangent to a line. (iii) the time when C_0, disc to the right of l, and C_i are tangent to a line.

The buffer zone of a valid right-right tangent l, for C_0 and C_i, may be affected by a disc to the right of l. We aim to maintain a convex chain of the discs to

the right of l, which contains a disc C_R with minimum distance to l in the direction perpendicular to $\overrightarrow{l}$. The disc C_R may contribute to the buffer zone. The buffer zone is the minimum time of the two values: (i) the time restricted by the Voronoi cells, (ii) the time when C_0, C_R and C_i are tangent to a line. For efficiency reasons, we cannot maintain the convex chain for l directly. In place of that, for each disc C_0, we divide the plane around it into wedges based on Voronoi neighbours of C_0. We show that a certain angular property is satisfied in these wedges and the number of wedges around a disc is proportional to the number of its Voronoi neighbours. The angular property helps us in computing convex chains and nearest neighbours for each valid right-right tangent within a wedge. The computation of buffer zones for valid right-right tangents is the most challenging part of the algorithm and the details are provided in Section 4.2. We also determine the time-intervals (buffer zones) for spiral segments after which time they can no longer be part of a shortest path. This comparatively simple step is omitted here. So, with each edge, we have an associated "expiry time" that the algorithm uses during execution. We summarize our result in the following theorem.

Theorem 1. *The shortest path among n growing discs problem can be solved in $O(n^2 \log n)$ time.*

Proof. The Voronoi diagram of a set of n discs can be computed in $O(n \log n)$ time, e.g., see [3]. By Lemma 8, right-right buffers can be computed in $O(n^2 \log n)$ time. By Lemma 9, right-left buffers can also be computed in $O(n^2 \log n)$ time. The graph has $O(n^2)$ nodes corresponding to departure curves, and $O(n^2)$ edges corresponding to the spiral segments and relevant valid tangents. Thus, using Dijkstra's algorithm on edges that have expiry times, a shortest path can be determined in $O(n^2 \log n)$ time. □

4 Computation of Buffer Zones

In this section, we discuss the key computation of buffer zones where we focus on right-right tangents in Section 4.2 and sketch right-left tangents in Section 4.3. We first describe, as preprocessing phase, the computation of valid tangents in Section 4.1.

4.1 Computation of Valid Tangents

We sketch how to find all valid tangents starting from disc C_0. First, we compute all tangents from C_0 to each disc and sort them in an angular order. Then we consider the discs in increasing order of distance from C_0, where, the distances are the lengths from the disc centers to O_0. Let C_i be the disc under consideration. We remove all the tangents in the sorted order that lie between l_{rr}^{0i} and l_{rl}^{0i}, and decide whether l_{rr}^{0i} and l_{rl}^{0i} are valid or not. If they are valid, then they have

not been removed before, and the tangent points must be inside the Voronoi cell of C_i. All the tangents around C_0 are stored in a balanced binary tree according to their slopes. Each deletion/insertion in the tree takes logarithmic time. The following lemma is easily obtained.

Lemma 4. *All valid tangents around a disc C_0 can be computed in $O(n \log n)$ time.*

4.2 Right-Right Buffer Zones

In this section, we describe the computation of the buffer zones for all valid right-right tangents $l_{rr}^{0j}, 1 \leq j \leq n$ of one disc, say $C_0 \in \mathscr{D} = \{C_0, C_1, \ldots, C_n\}$. The section is organized as follows: we first show which discs constrain the buffer zone of a valid l_{rr}^{0i}. Then, we present how to compute the buffer zones for all valid $l_{rr}^{0j}, 1 \leq j \leq n$, when C_0 is point ($r_0 = 0$). We finish by showing how to compute the buffer zones when the radius $r_0 > 0$. Since all computations of the buffer zones are for valid right-right tangents, we may assume that all right-right tangents used in this section are valid.

Let l be a right-right tangent. We denote by $P^R(l)$ the set of discs that are to the right of l. We will define the predecessor disc of a valid tangent line l_{rr}^{0i}. Without loss of generality, we assume that the slope of l_{rr}^{0i} is 90^o. A disc $C_j \in P^R(l_{rr}^{0i})$is a predecessor disc of l_{rr}^{0i} if slope(l_{rl}^{0j})= max(slope$(l_{rl}^{0k})), C_k \in P^R(l_{rr}^{0i})$. We denote by Pred$(l_{rr}^{0i})$ the predecessor disc of l_{rr}^{0i}.

One of these discs affects the maximum value of each buffer zone associated with l_{rr}^{0i} as given in the next observation.

Observation 1. *The maximum value of the buffer zone associated with l_{rr}^{0i} is determined by one of these discs:*

1. *a disc that is a Voronoi neighbour of C_0, or C_i, or*
2. *disc $C_k \in P^R(l_{rr}^{0i})$ with the minimum distance to l_{rr}^{0i}.*

Determining the effect of Voronoi neighbours on the buffer zone is easy (see Lemma 3). Thus, we focus on finding the disc C_k defined in Observation 1 case 2.

A brute force approach for determining C_k is as follows: evaluate the distance of each disc in $P^R(l_{rr}^{0i})$ and select the disc with the minimum distance. However, this approach would take $O(n)$ time for each $l_{rr}^{0j}, 1 \leq j \leq n$. Thus, yielding $O(n^2)$ time to determine the buffer zones associated with $l_{rr}^{0j}, 1 \leq j \leq n$. This would result in an $O(n^3)$ time algorithm for computing the buffer zones associated with each $l_{rr}^{ij}, 1 \leq i, j \leq n, i \neq j$.

Next, we show how to compute C_k efficiently. We start by defining a convex chain of discs. We omit from the definition of the convex chain, the degenerate cases where the convex chain only consists of one, or two discs.

Definition 1. *An ordered set of discs $U = \{C_1, \ldots, C_m\}, U \subseteq P^R(l)$ forms a convex chain along l, if (i) the projections of the disc centers onto l appear in the order $O_1, \ldots, O_m$, (ii) the CCW angle from $l_{ll}^{(i-1)i}$ to $l_{ll}^{i(i+1)}$, $2 \leq i \leq m-1$,*

does not exceed 180° *and (iii) the concave opening of the convex chain is to the right of* l.

A convex chain X, as defined in Definition 1, is not merely a chain that consists of an alternating sequence of discs and tangents between the discs. The convex chain X is constructed with respect to time. Let T_1 be a time associated with C_1. The line segment which connects C_1 to C_2 in X is not a tangent in the standard mathematical sense, but rather is a time-dependent tangent, l_{ll}^{12}, between C_1 and C_2 as defined in Section 2. Thus, l_{ll}^{12} determines the time associated with C_2, which is denoted by T_2. When constructing l_{ll}^{23} we use the radius of C_2, at time T_2. In general, when constructing $l_{ll}^{i(i+1)}$ we use the radius of C_i, at time T_i.

Next we show how to use a convex chain to determine the maximum value of the buffer zone associated with l_{rr}^{0i}.

Lemma 5. *Let X be a convex chain such that all discs of $P^R(l_{rr}^{0i})$ are contained in the concave part of X. The extreme disc, C_j of X with respect to l_{rr}^{0i} has the smallest distance to l_{rr}^{0i}.*

Another important property of the convex chain which is used to determine the buffer zone associated with l_{rr}^{0i} is given in the following lemma.

Lemma 6. *Let X be a convex chain such that all discs of $P^R(l_{rr}^{0i})$ are contained in the concave part of X and let $c = Pred(l_{rr}^{0i})$. The convex chain X contains c.*

The following observation immediately follows from Lemmas 5 and 6.

Observation 2. *Let $X = C_1, \ldots, C_m$ be a convex chain such that all discs of $P^R(l_{rr}^{0i})$ are contained in the concave part of X and let $C_j \in X = Pred(l_{rr}^{0i})$. The extreme disc with respect to l_{rr}^{0i} is either the predecessor disc C_j, or the extreme disc of the discs appearing before C_j, i.e, $C_1, \ldots, C_{j-1}$.*

So far, we have established several properties and relationships between l_{rr}^{0i} and the disc that determines the buffer zone of l_{rr}^{0i}. We divide the plane into four quadrants, Q_1, Q_2, Q_3, and Q_4, around C_0 with origin $O_0 = (0, 0)$, listed in CCW order, with Q_1 being the North-East quadrant. A corollary from Observation 1 is that the disc that determines the buffer zone of l_{rr}^{0i} in Q_1 is either in Q_1 or in Q_4. Therefore, we must ensure that to determine the buffer zones of the discs in Q_1 we must also consult the discs in Q_4. We do so by constructing convex chains for the discs in Q_4 which are termed external convex chains. Once the external convex chains are constructed, a buffer zone of $l_{rr}^{0i} \in Q_1$ is determined by finding predecessor discs and then determining the extreme disc in each of the associated convex chains.

First, we describe our algorithm for the case where C_0 is a point and then generalize it to the case where C_0 is a disk. We show how to compute the buffer zones of all disks $C_j \in \mathscr{D}$ such that l_{rr}^{0j} is in the North-East quadrant, Q_1.

Algorithm 1. *Compute buffer zones when C_0 is a point.*

1. *Sort all l_{rl}^{0j} $1 \leq j \leq n$ by slope(l_{rl}^{0j}).*
2. *Incrementally build convex chains by adding one disc at a time in order of slopes. Let l_{rl}^{0j} be the current slope to be processed. Disc C_j is a candidate for being a predecessor disc. Since it has the largest slope among the discs considered so far, it must appear on the convex chain constructed so far. Moreover, we need to keep only the part of the convex chain where C_j is the last disc (Observation 2). We try to insert C_j as the last disc of the current chain. If this preserves the convexity of the chain then we are done. Otherwise, we delete the last disc from the current convex chain and repeat this process until C_j can be successfully inserted into the current convex chain as last element. Figure 3 provides an illustration.*
3. *By adapting the algorithm of [1] determine Pred(l_{rr}^{0i}), $1 \leq i \leq n$.*
4. *Incrementally compute the external convex chains of the discs in Q_4.*
5. *Sort all l_{rr}^{0i} $1 \leq j \leq n$ by slope(l_{rr}^{0i}).*
6. *Compute the buffer zone for each l_{rr}^{0i} in sorted order of slopes(l_{rr}^{0i}):*
   (a) *Find the predecessor disc in Q_1.*
   (b) *Find the disc in Q_4 such that it projects to l_{rr}^{0i} and is extreme in the direction perpendicular to l_{rr}^{0i}.*
   (c) *Compute the extreme disc in the convex chain associated with the predecessor disc found in Step 6a.*
   (d) *Set the buffer zone with respect to the disc with minimum distance to l_{rr}^{0i} as computed in Steps 6b and 6c. Next, take as buffer zone, the minimum of this value and the values T_p and $T_{p'}$ as defined in Lemma 3.*

Lemma 7. *The buffer zone for each valid right-right tangent can be computed in $O(n \log n)$ time inside each quadrant when C_0 is a point using Algorithm 1.*

Now consider the case when C_0 is a disc with non-zero radius and therefore C_0 has a curvature. To overcome the associated difficulties, we modify our partitioning scheme. Instead of partitioning into quadrants, we use neighbours of disc C_0 in the Voronoi diagram of the discs in $\mathscr{D}$ to partition the plane into *wedges*. Depending on the size of a wedge and the arrangement of the discs within a wedge, we may need to further partition the wedges into sub-wedges. This subdivision is based on the empty circle property of Voronoi diagrams of discs and results in each wedge being further divided into at most three sub-wedges. Once the plane is partitioned, each resulting wedge allows us to apply Algorithm 1. This step is fairly technical and details will be provided in the full version of this paper. We now present the algorithm:

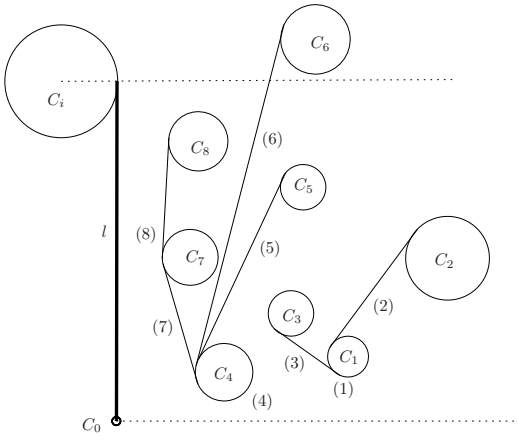


Fig. 3. Incrementally construct convex chains. Disc C_1, ..., C_8 are sorted in order of slopes of its right-left tangents from C_0 (point). E.g., convex chain (1): C_1; convex chain (2): C_1, C_2; convex chain (4): C_4; and convex chain (8): C_4, C_7, C_8. The growth of the discs is not shown in the figure.

Algorithm 2. *Compute buffer zones when C_0 is a disc.*
(1) Find Voronoi neighbours of C_0. Let C_A and C_B be two Voronoi neighbours of C_0 such that they are adjacent.
(2) Divide the plane into wedges around C_0. The boundary edge L_1 (resp. L_2) of a wedge passes through the centers of C_0 and C_A (resp. C_B).
(3) Refine the partitioning of the wedges into at most three sub-wedges.
(4) In each wedge, execute Algorithm 1.

Lemma 8. *The buffer zones for all the valid right-right tangents in the plane can be computed in $O(n^2 \log n)$ time.*

Proof. In Algorithm 2, the number of wedges for disc C_0 can be bounded by the number of its Voronoi neighbours. Within each wedge, we spend $O(n \log n)$ time using Algorithm 1. □

4.3 Right-Left Buffer Zones

Lastly, because of space constraints, we can only sketch our algorithm for computing buffer zones of right-left departure curves on disc C_0. Two discs may have effects on the buffer zone of a right-left tangent l. One is to the right of l, say C_R, and the other one is to the left of l, say C_L.

Let l be a right-right (right-left) tangent. The first tangent l', either to the left of l or to the right of l, with center projection of the corresponding disc center onto l outside of l, can be computed in $O(\log n)$ time, after $O(n \log n)$

Algorithm 3 *Compute-right-left-buffer-on-Disc-C_0*
Input: *Disc set $\mathscr{D}$, $C_0 \in \mathscr{D}$, all tangents.*
Output: *$T_r(l_{rl}^{0i})$, for all valid tangents l_{rl}^{0i} (as computed in Section 4.1).*
For each valid right-left tangent l_{rl}^{0i},
(1) Find the disc, C_L, (if it exists) to the left of l_{rl}^{0i}, where, C_0, C_L and C_i determine $T_r(l_{rl}^{0i})$: l_{rr}^{0L} has the smallest slope larger than the slope of l_{rl}^{0i} among all the l_{rr}^{0k}, where, C_k is to the left of l_{rl}^{0i}. Compute the start time, T', on C_0 for the robot traveling along the line tangent to C_0, C_L and C_i.
(2) Find the disc, C_R, (if it exists) to the right of l_{rl}^{0i}, where, C_0, C_R, and C_i determine $T_r(l_{rl}^{0i})$: l_{ll}^{Ri} is the first left-left tangent to C_i to the right of l_{rl}^{0i}. Compute the start time, T'', on C_0 for the robot traveling along the line tangent to C_0, C_R, and C_i.
(3) Determine T_p and $T_{p'}$ by using Lemma 3.
(4) $T_r(l_{rl}^{0i}) = \min\{T_p, T_{p'}, T', T''\}$.

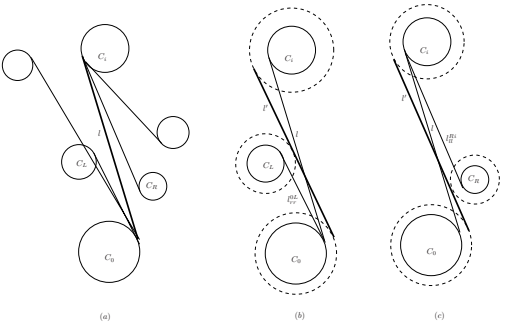


Fig. 4. (*a*) The buffer zone of the right-left tangent l may be affected by C_L to the left of l, or C_R to the right of l. (*b*) The new tangent to C_0, C_L and C_i, shown in bold. (*c*) The new tangent to C_0, C_R and C_i, shown in bold.

preprocessing time. This is done by adapting the solution to all-nearest-smaller-value problem described in [1] to our scenario (details are available in the full version of this paper).

Lemma 9. *The buffer zone of each valid right-left tangent can be computed in $O(\log n)$ time. All buffer zones for right-left tangents can be determined in $O(n^2 \log n)$ time.*

5 Conclusions

The result of this paper is an algorithm to compute a collision-free shortest path between two points among a set of n discs growing at the same speed in $O(n^2 \log n)$ time. It is an interesting problem to consider the scenario when the discs grow at different speeds.

References

1. Berkman, O., Schieber, B., Vishkin, U.: Some doubly logarithmic optimal parallel algorithm based on finding all nearest smaller values. Technical Report, University of Maryland, UMIACS-TR-88-79 (1988)
2. Chang, E.C., Choi, S.W., Kwon, D.Y., Park, H., Yap, C.K.: Shortest path amidst disc obstacles is computable. In: Proc. of the 21st annual Symposium on Computational Geometry (SoCG), pp. 116–125 (2005)
3. Jin, L., Kim, D., Mu, L., Kim, D-S., Hu, S.-M.: A sweepline algorithm for Euclidean Voronoi diagram of circles. Computer-Aided Design 38, 260–272 (2006)
4. Kim, D.S., Yu, K., Cho, Y., Kim, D., Yap, C.: Shortest path for disc obstacles. In: Laganà, A., Gavrilova, M., Kumar, V., Mun, Y., Tan, C.J.K., Gervasi, O. (eds.) ICCSA 2004. LNCS, vol. 3045, pp. 62–70. Springer, Heidelberg (2004)
5. Mitchell, J.S.B: Geometric shortest paths and network optimization. In: Sack, J.-R., Urrutia, J. (eds.) Handbook of Computational Geometry, pp. 633–701. Elsevier, Amsterdam (2000)
6. van den Berg, J.: Path planning in dynamic environments. Ph.D. Thesis, Utrecht University, Utrecht, The Netherlands (2007)
7. van den Berg, J., Overmars, M.: Planning the shortest safe path amidst unpredictably moving obstacles. In: Proc. Workshop on Algorithmic Foundations of Robotics (2006)

Optimal Triangulation with Steiner Points

Boris Aronov[1], Tetsuo Asano[2], and Stefan Funke[3]

[1] Polytechnic University, Brooklyn, NY, USA
[2] School of Information Science, JAIST, Japan
[3] Max-Planck-Institut für Informatik, Saarbrücken, Germany

Abstract. There are many ways to triangulate a simple n-gon; for certain optimization criteria such as maximization of the smallest internal angle it is known how to efficiently compute the best triangulation with respect to this criterion. In this paper we consider a natural extension of this problem: Given a simple polygon P and one Steiner point p in its interior, determine the optimal location of p and a triangulation of P and p which is best amongst all triangulations and placements of p. We present a polynomial-time algorithm for this problem when the optimization criterion is maximization of the minimum angle. Furthermore, we also provide a more general polynomial-time algorithm for finding the optimal placement of a constant number of Steiner points under the same optimization criterion.

1 Introduction

Triangulations of simple polygons arise in many applications. Some triangulation of a given simple polygon can even be computed in linear time using Chazelle's algorithm [6]. Optimizing some criterion over all triangulations is also possible. For example, a popular optimization criterion is to maximize the minimum angle of any triangle. Such a triangulation is known as a *constrained Delaunay triangulation*; it can be obtained in $O(n \log n)$ time for an n-gon [7]. We could also find a *minimum-weight triangulation* that minimizes the total length of chords required for triangulation using dynamic programming. Dynamic programming is also powerful enough to find a triangulation in which the worst aspect ratio of resulting triangles is minimized, where the *aspect ratio* of a triangle is the ratio of length of the longest side to its *width*, i.e., its smallest height.

In this paper we are interested in what happens when we allow one Steiner point in the triangulation. More precisely, given a simple polygon P, we want to find a point p in the interior of P such that the quality of the optimal triangulation of $P+\{p\}$ is optimized under a given optimization criterion. If maximization of the minimum internal angle is the goal, we want to find a location of an interior point p such that the minimum angle of the optimal triangulation of $P+\{p\}$ is maximized among all possible interior points p. As far as the authors know, there is no previous study of the question. Our main concern in this paper is to develop a polynomial-time algorithm.

A natural extension of this problem is to allow for more Steiner points to be inserted or to use different optimization criteria for the triangulation. The

T. Tokuyama (Ed.): ISAAC 2007, LNCS 4835, pp. 681–691, 2007.

extension to multiple Steiner points is not trivial at all. In fact, it seems no simple algorithm exists for finding an optimal set of Steiner points. We present a fairly involved polynomial-time algorithm that optimally places any constant number k of Steiner points. Using a different optimization criterion is also interesting. Minimization of the largest internal angle, minimization of the largest slope, and minimization of the largest aspect ratio are rather popular criteria [3,4], but no $O(n \log n)$ algorithm is known for these criteria except for that of maximization of the smallest angle (if adding Steiner points is not permitted). So, although it is challenging to extend our ideas to other criteria, in this paper we shall only consider the maxmin angle criterion for which we can design polynomial-time algorithms for the case of one or a constant number of Steiner points.

This problem is closely related to mesh improvement. Given a triangulation of some bounded domain, we sometimes want to improve the quality of the triangulation by relocating internal vertices (we assume internal vertices can be moved while vertices on the domain boundary are fixed). In the so-called Laplacian method (see, e.g., [9]) an internal vertex is relocated to the barycenter of the polygon defined by its incident triangles. It works well in practice, but the barycenter is not always the best location for a vertex. We want to emphasize that in the Laplacian method the *topology* of the triangulation, that is its underlying graph, is unchanged. Hence the barycenter is just a candidate for a good location when the topology is fixed. It is not known what a good location for the vertex is when the topology is allowed to change. Naturally one might expect better triangulations when topology changes are allowed.

Further closely related results are known in the literature under the heading of *Delaunay refinement*. Here one is given a planar straight line graph (PSLG) represented by a set of vertices and non-intersecting edges, and the goal is to triangulate this PSLG using 'fat' triangles, the latter being important if the obtained subdivision is used for example in finite element method calculations. 'Fatness' can be achieved by maximizing the smallest angle in the computed triangulation. Delaunay refinement algorithms repeatedly insert Steiner points until a certain minimum angle is achieved; the major goal here is to bound the number of necessary Steiner points. In some way our paper approaches the same problem from the opposite direction: how much can we improve the 'fatness' of our triangulation with few Steiner points. See [14] for an excellent survey on Delaunay refinement techniques.

This paper is organized as follows: Section 2 describes a polynomial-time algorithm for computing the optimal location for a single Steiner point. The more general case of a constant number of Steiner points is considered in Section 3. Section 4 includes conclusions and future work.

2 Triangulation Using One Steiner Point

The main problem we address in this section is the following:

Problem 1. Given a simple n-gon P, find a triangulation of the interior of P with one Steiner point maximizing the smallest internal angle.

To that end we will consider the following more general problem:

Problem 2. Given a set of points X and a set of non-crossing edges E with endpoints in X. Find a triangulation of X and one Steiner point which respects the edges in E and maximizes the smallest internal angle.

As we will see below, our solution to Problem 2 also provides us with a solution to Problem 1.

Before proceeding, we need several definitions. Given a set of points X and a set of non-crossing edges E with endpoints in X, we say $a \in X$ *sees* $b \in X$ iff the line segment ab does not cross any edge in E (note that, for a line segment $ab \in E$, a sees b). Point a is *visible* to a set Y if a can be seen from some point in Y.

Definition 1. *The* constrained Delaunay triangulation (CDT) *of a set of points X and a set of non-crossing edges E with endpoints in X contains exactly those edges (a, b), $a, b \in X$ for which either $(a, b) \in E$, or (1) a sees b and (2) there exists a circle through a and b such that no $c \in X$ contained in the interior of the circle is visible from ab.*

A constrained Delaunay triangulation $\mathrm{CDT}(X, E)$ for (X, E) can be computed in $O(n \log n)$ time and for non-degenerate (free of cocircular quadruples of points) point sets forms a proper triangulation, i.e., a decomposition of the convex hull of X into triangles. It maximizes the minimum interior angle of any triangulation of (X, E) that uses only the points of X as triangulation vertices; in fact, the CDT lexicographically maximizes the list of angles from smallest to largest, see [5] for an extensive list of references.

In the following we are interested in how the constrained Delaunay triangulation changes when some point p is added to X. Definition 1 implies that the circumcircle of $\triangle abc \in \mathrm{CDT}(X, E)$ cannot contain a vertex other than a, b, c visible from the interior of $\triangle abc$. Hence the insertion of p can only affect triangles in $\mathrm{CDT}(X, E)$ in the circumcircle of which p lies. More precisely, we say an edge $e \notin E$ is *invalidated* by p iff p lies in the intersection of the circumcircles of the two adjacent triangles of e and p is visible from the interior of both triangles (for an edge on the convex hull of X consider the artificial triangle with one vertex at infinity and the corresponding 'circumhalfplane').

Lemma 1. *$\mathcal{D}(p) := \mathrm{CDT}(X \cup \{p\}, E)$ can be obtained from $\mathcal{D} := \mathrm{CDT}(X, E)$ by deleting all edges in $\mathrm{CDT}(X, E)$ invalidated by p and retriangulating the resulting 'hole' H in a star fashion from p.*

Proof Clearly all edges in $\mathcal{D}$ not invalidated by p are part of $\mathcal{D}(p)$ according to Definition 1. Furthermore, it is not possible that $\mathcal{D}(p)$ contains an edge vw which was not present in $\mathcal{D}$, with $v, w \neq p$, since the insertion of p only decreases the number of admissible edges on the original vertices. Therefore, new edges in $\mathcal{D}(p)$ have to have p as one endpoint. Connecting p to all visible vertices of H is the only way to obtain a triangulation again. □

Consider the arrangement $\mathcal{A}$ defined by the triangles of $\mathcal{D}$ and their circumcircles. (For a technical reason explained below, we further refine $\mathcal{A}$ into constant-size "trapezoids," by replacing it with its trapezoidal decomposition [2, p. 124].) We argue that the topology of $\mathcal{D}(p)$ does not change when p is moved within one cell σ of this arrangement.

Lemma 2. *If a point $p \in \sigma$ can be seen from the interior of a triangle $\triangle abc \in \mathcal{D}$ whose circumcircle contains p, then all points within σ can be seen from the interior of $\triangle abc$.*

Proof Let x be a point in $\triangle abc$ which sees p and assume there exists a point $p' \in \sigma$ which cannot be seen from x. Consider the line segment xq as q moves towards p' along pp'. If p' cannot be seen from x, at some point xq must hit a line segment $e \in E$ obstructing the view, at an endpoint of e. Hence this endpoint must be visible from x and must lie in the interior of the circumcircle of $\triangle abc$, contradicting $\triangle abc \in \mathcal{D}$. □

Since all points within a cell σ also lie within the same set of circumcircles, we have shown that all points within σ of the above arrangement invalidate the same set of edges. It remains to show that all points p within this cell σ behave the same in terms of visibility from vertices of triangles of which at least one edge was invalidated by p.

Lemma 3. *Let $e = (b, c)$ be an edge of $\mathrm{CDT}(X, E)$, $\triangle abc$ and $\triangle bcd$ the respective adjacent triangles to e. If e is invalidated by p then p sees a, b, c and d.*

Proof Let $x \in \triangle abc$ be visible from p and suppose, without loss of generality, that x cannot see a. Consider the line segment $\overline{yp}$ as y moves along $\overline{xa}$ towards a. At some point $\overline{xp}$ must meet a constraining edge $e \in E$ at a vertex of X lying in the interior of the circumcircle of $\triangle abc$, contradicting the assumption that $\triangle abc \in \mathcal{D}$. □

We have shown that all points within a cell σ behave identically in terms of invalidation of edges as well as visibility hence leading to the identical topology of $\mathcal{D}(p)$. Furthermore observe that the complexity of $\mathcal{A}$ is $O(n^2)$ since the arrangement of $O(n)$ circles has complexity $O(n^2)$ and the additional $O(n)$ line segments only add $O(n^2)$ intersection points. We summarize our findings in the following corollary.

Corollary 1. *The arrangement $\mathcal{A}$ defined by the triangles of $\mathcal{D}$ and their circumcircles characterizes the different topologies of $\mathrm{CDT}(X \cup \{p\}, E)$ after insertion of a Steiner point p in the sense that all placements of p within the same cell of $\mathcal{A}$ lead to the same topology. The size of $\mathcal{A}$ is $O(n^2)$.*

We note that the arrangement will in general be overrefined in a sense that points in different cells of $\mathcal{A}$ might lead to the same topology of $\mathcal{D}(p)$.

For our original Problem 1 of triangulating a polygon P with one Steiner point p we compute the arrangement $\mathcal{A}$ with respect to $\mathrm{CDT}(X, E)$ where X are the polygon vertices and E the polygon edges, and only consider the cells of $\mathcal{A}$ inside P.

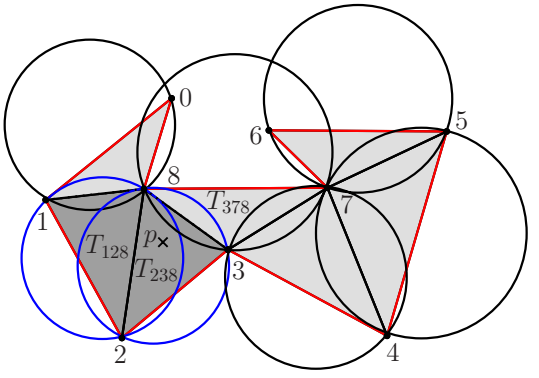


Fig. 1. A point p in the interior of a simple polygon is contained in the circumcircles of the two Delaunay triangles T_{128} and T_{238}, but not in that of T_{378}

Fig. 1 shows an example. The interior of the polygon (shaded) is partitioned into cells by all triangles and their respective circumcircles. The circumcircles of the triangles T_{128} and T_{238} (shaded darker) contain the point p and p is visible from both triangles. So, edge $\overline{28}$ cannot be included in the constrained Delaunay triangulation after insertion of p. On the other hand, p lies outside the circumcircle of T_{378} and thus this triangle is left unchanged after the insertion.

We have seen that when a point p is placed somewhere within a cell $\sigma \in \mathcal{A}$, a fixed set of edges is invalidated, producing a star-shaped 'hole' $H = H(\sigma)$ in $\mathrm{CDT}(X, E)$. We then optimize the minimum angle in the triangulation, over all possible placements of $p \in \sigma$, only focusing on the interior angles in the star triangulation of H, as the rest of the triangulation is unaffected by the insertion of p.

Algorithm for finding an optimal location of a Steiner point
Input: a set X of points and a set of non-crossing edges E with endpoints in X

1. Compute the constrained Delaunay triangulation $\mathcal{D} := \mathrm{CDT}(X, E)$.
2. Construct the arrangement induced by all triangles of $\mathcal{D}$ and their circumcircles. Refine it by a trapezoidal decomposition to obtain $\mathcal{A}$.
3. For each cell σ of $\mathcal{A}$:
   - Determine the set of edges invalidated by any Steiner point in σ and remove them to form the hole H
   - Compute an angular Voronoi diagram for H, truncated to within σ.
   - For each Voronoi edge in the truncated Voronoi diagram, find a point maximizing the minimum angle along the edge.
   - For each connected component of a boundary edge of the cell σ lying in the same cell of the truncated Voronoi diagram, find a point maximizing the minimum angle along this curve.
4. Return the triangulation yielding the best angle found.

The next section will describe in detail how to actually treat a cell $\sigma \in \mathcal{A}$ using angular Voronoi diagrams. Note that the above algorithm also solves our

original Problem 1 of optimizing the triangulation of a simple polygon P using one Steiner point: In step 3 of the algorithm we only need to consider cells that lie in the interior of P and when maximizing the minimum angle we also only consider triangles that lie in the interior of P.

Finding an optimal triangulation with fixed topology. It remains to find an optimal point p within a specified cell σ bounded by circles and lines within a given star-shaped polygon $H := H(\sigma)$ that maximizes the smallest interior angle in the star-triangulation of H from p.

Given a star-shaped polygon H, we can find an optimal point p that maximizes the smallest *visual angle* from p to all edges of H, i.e., the angle at which any edge of H is seen from p. Matoušek, Sharir and Welzl [12] gave an almost linear-time algorithm within the framework of LP-type problems. Asano et al.[1] also gave efficient algorithms for the same problem using parametric search or the so-called angular Voronoi diagram. Our question is slightly different. It is not enough to maximize the smallest visual angle around the point p to be inserted: the smallest internal angle may be incident to the boundary of H rather than p! Another difficulty is that we want to find an optimal point p constrained to lie in σ, a cell in the arrangement $\mathcal{A}$ bounded by circular arcs and straightline segments, rather than ranging over all of H. It seems to be hard to adapt the aforementioned algorithms based on LP-type problem formulation or parametric search for this purpose, but fortunately the one using the angular Voronoi diagram can be adapted here.

The *angular Voronoi diagram* for a star-shaped polygon H is defined as a partition of the plane according to the polygon edge that gives the smallest visual angle [1]. A point p belongs to the Voronoi region of a polygon edge e if the visual angle from p to e is smaller than that to any other polygon edge of H. It is known that it consists of straight line segments or curves of low degree and has total complexity $O(n^{2+\varepsilon})$, for any $\varepsilon > 0$ and with implied constant depending on ε.

We have to modify the definition of the angular Voronoi diagram to take into account the angles associated with polygon edges as well. Given a star-shaped polygon H as a set of its bounding edges $\{e_0, e_1, \ldots, e_n := e_0\}$ and a point p in the plane, for each edge e_i we form the triangle $Tr(p, e_i)$ by connecting the endpoints to p. The value $f(p, e_i)$ is defined to be the smallest internal angle in the triangle $Tr(p, e_i)$. The region $\mathrm{Vor}(e_i)$ of an edge e_i is defined by a set of points p at which $f(p, e_i)$ is smallest among all edges, that is,

$$\mathrm{Vor}(e_i) := \{p \in \mathbb{R}^2 \mid f(p, e_i) \leq f(p, e_j) \quad \forall e_j \in H\}.$$

Given a line segment ab, we partition the plane into four regions by two circles C_a and C_b centered at a and b, respectively, with the radius $|ab|$ and the perpendicular bisector l_{ab} of ab. Refer to Fig. 2. If the point p lies to the left of the line l_{ab} (more precisely, the halfplane defined by the line l_{ab} that contains the point a) and in the exterior of C_b, then the smallest internal angle of $\triangle abp$ is $\angle apb$. If p lies in the same halfplane but in the interior of C_b, then $\angle abp$ is smallest. The smallest angle in the right halfplane is similarly defined. Fig. 2 illustrates three

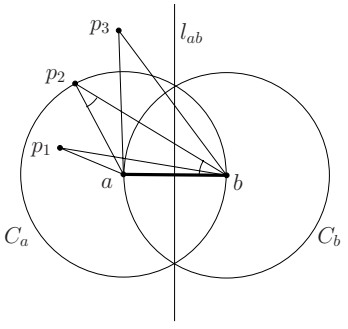


Fig. 2. Partition of the plane into regions according to which angle of $\triangle abp$ is smallest

different situations, with points p_1, p_2, p_3 lying outside, on the boundary of, and inside C_a, respectively.

Fig. 3 in which Voronoi regions are painted by colors associated with polygon edges gives an example of such a modified angular Voronoi diagram. The given polygon H is indicated by solid white lines.

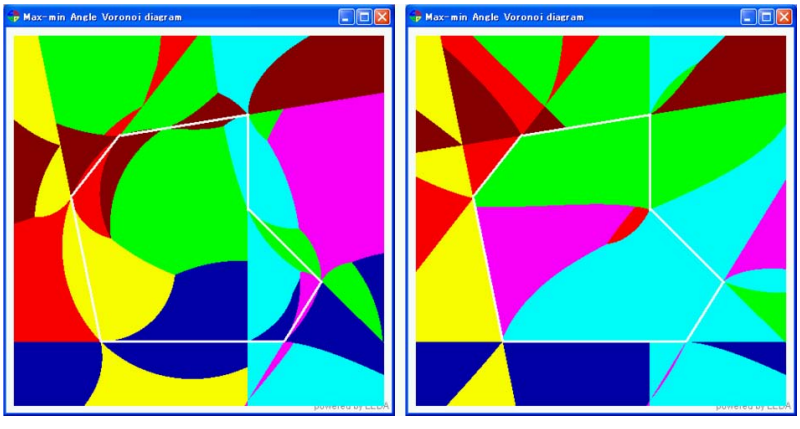


Fig. 3. Modified angular Voronoi diagram (left) and original angular Voronoi diagram

Once we construct the modified angular Voronoi diagram, we can look for an optimal placement for p at Voronoi vertices, along Voronoi edges, or along the boundary of σ just like in the original angular Voronoi diagram [1]; it is easy to see that the maximum does not occur in the interior of Voronoi regions. Recall that we have refined $\mathcal{A}$ so that a cell $\sigma \in \mathcal{A}$ is a constant-complexity region. In particular, each function $f(p, e_i)$, viewed as truncated to within σ, is a well-behaved function with a constant-complexity domain. Hence standard results of envelope theory [13] imply that the modified angular Voronoi diagram truncated to within σ has at most $O(n^{2+\varepsilon})$ edges, including connected portions of Voronoi edges within σ and portions of Voronoi cells lying along the boundary of σ. The

computations can be performed in time $O(n^{2+\varepsilon})$ per cell, for a total of $O(n^{4+\varepsilon})$, since cell processing dominates the runtime of the algorithm.

3 Triangulation Using Several Steiner Points

We now turn our attention to the situation when two Steiner points, p and q, are permitted to be placed in a simple n-gon P. We start with the triangulation $\mathcal{D} := \mathrm{CDT}(X, E)$. We consider the space $P^2 := P \times P$ of all possible placements of the two points. We aim to identify the best placement of p, q in order to maximize the smallest angle in the resulting constrained Delaunay triangulation $\mathcal{D}(p, q) := \mathrm{CDT}(X \cup \{p, q\}, E)$ (where as before X is the set of vertices of P and E is the set of its edges). As in the previous section, we partition P^2 according to the topology of $\mathcal{D}(p, q)$, then use an analog of the modified angular Voronoi diagram from previous section to determine which angle is smallest in the triangulation for every choice of $(p, q) \in P^2$ and search the resulting diagrams for the placement maximizing the minimum angle. This plan is complicated by the need to explicitly identify all possible triangulation topologies. Instead, we will arrive at this partition indirectly, as detailed below.

In this section, we focus on constructing a polynomial-time algorithm, without any attempt at optimizing the running time. Such an optimization might be a good topic for further research, especially when coupled with some heuristics to eliminate infeasible placements of p and q in order to reduce the search space, which we have developed but have been unable to include in this version due to space limitations.

We first recall a standard fact, the analogue of Definition 1 [7].

Fact 1. *A triangle $\triangle abc$, for $a, b, c \in X$ is present in $\mathrm{CDT}(X, E)$ if and only if a, b, c are pairwise visible and no other vertex of E visible from any point in $\triangle abc$ lies in the circumcircle of $\triangle abc$.*

Consider $\mathcal{D}(p, q)$ as defined above and consider a potential triangle $\triangle abc$ in it. Let $f(a, b, c; p, q)$ be a partial function defined as follows: it is defined for $(p, q) \in P^2$ if and only if $\triangle abc$ is present in $\mathcal{D}(p, q)$ and the value of f is the measure of $\angle abc$. Then clearly the smallest angle in $\mathcal{D}(p, q)$ is

$$m(p, q) := \min_{(a,b,c)} f(a, b, c; p, q),$$

where the minimum is taken over all triples of distinct elements in $X \cup \{p, q\}$, for which the function $f(a, b, c; p, q)$ is defined at (p, q). The desired triangulation maximizing the minimum angle is just the maximum of function $m(p, q)$ over all of P^2.

We would like to apply envelope theory to compute $m(p, q)$ for all p, q. In a typical lower envelope argument [13], however, functions are well-behaved (e.g., algebraic of bounded degree) *and* defined over domains of constant description complexity (say, by a constant-length semialgebraic condition of bounded degree). In our case, the form of functions f is simple enough (if one uses, for

example, $\cos^2 \angle abc$ to compare angles, to avoid transcendental functions, this is a low-degree rational function of the coordinates of the points).

The difficulty is in their domains of definition—they are in general not of constant complexity and hence the envelope analysis is not applicable directly. We instead decompose P^2 into constant-size cells in such a manner that each function is either total, or totally undefined on every cell c of the decomposition.

In order to construct such a decomposition, we observe that the boundaries of the domain of definition of a function $f(a, b, c; p, q)$ are given precisely by Fact 1. Namely, if we view p and q as moving, a triangle $\triangle abc$ formed by three points from among the vertices of P and/or p, q can cease to belong to $\mathcal{D}(p, q)$ only when some visibility constraint is violated (two possibilities: a vertex becomes collinear with pq, or p or q becomes collinear with a line defined by two vertices) or when a cocircularity is created or destroyed (again, two possibilities: p or q becomes cocircular with three vertices, or both p and q become cocircular with two vertices). All four possibilities correspond to a low-degree hypersurface in $\mathbb{R}^4$, and the number of possibilities is clearly polynomial, since there are n vertices in all.

We collect all these hypersurfaces, add boundaries of P^2 to the arrangement, and truncate it to within P^2. We then refine the resulting partition of P^2 to contain only constant-size cells (e.g., via a cylindrical algebraic decomposition or a vertical decomposition [8,10]); the resulting decomposition $\mathcal{A}$ is still of polynomial size.

$\mathcal{A}$ is a subdivision of P^2 into polynomial number of constant-complexity cells σ with the property that each function $f(a, b, c; \cdot, \cdot)$ is either defined on the entire cell σ or undefined on all of σ. Functions are algebraic of bounded degree. There is a polynomial number of functions. Hence standard envelope theory [13] concludes that the minimization diagram (i.e., the decomposition of space into maximal connected portions over each of which a fixed function or set of functions achieves the pointwise minimum) of this collection of functions can be computed and, if necessary, further refined to constant-complexity cells, in polynomial time. Over each cell of the minimization diagram, a single function appears on the lower envelope, so we can determine in constant time its largest value. Taking the maximum over all cells, we obtain the placement of $p, q \in P$ that maximizes the minimum angle of any triangulation of P with two Steiner points p, q, in polynomial time, as promised.

We summarize our findings in the following theorem.

Theorem 2. *Given a polygon P with n vertices, or more generally a set of n points X and a collection E of non-crossing edges connecting them, we can find two points p and q such that the minimum angle of the constrained Delaunay triangulation* $\mathrm{CDT}(X \cup \{p, q\})$ *(and thus of any other triangulation with vertices $X \cup \{p, q\}$ respecting E) is maximized in time polynomial in n.*

The same method applies almost verbatim to any constant number of Steiner points. We omit the details due to space limitations.

4 Conclusions and Future Work

In this paper we have presented polynomial-time algorithms for finding optimal placement of one, or a constant number of, Steiner points to be inserted in a simple n-gon to maximize the minimum internal angle of triangulation. It would be interesting to improve the dependence of the latter algorithm on the number of Steiner points, to construct practical algorithms for solving the problems, and to extend our analysis to other optimization criteria.

Acknowledgments

Work by B.A. has been partially supported by a grant from the U.S.-Israel Binational Science Foundation and by NSA MSP Grant H98230-06-1-0016. Work by T.A. was partially supported by the Ministry of Education, Science, Sports and Culture, Grant-in-Aid for Scientific Research on Priority Areas and Scientific Research (B). Part of the work of T.A. was performed while he was visiting Max-Planck Institute für Informatik, Germany, Carleton University, Canada, and New York University, USA, all in 2006. Work by S.F. was partially supported by the Max Planck Center for Visual Computing and Communication (MPC-VCC) funded by the German Federal Ministry of Education and Research (FKZ 01IMC01).

References

1. Asano, T., Katoh, N., Tamaki, H., Tokuyama, T.: Angular Voronoi Diagram with Applications. In: Proc. International Symposium on Voronoi Diagram in Science and Engineering, Banff, Canada, pp. 32–39 (2006)
2. de Berg, M., van Kreveld, M., Overmars, M., Schwarzkopf, O.: Computational Geometry: Algorithms and Applications, 2nd edn. Springer, Berlin (2000)
3. Bern, M.: Triangulations and Mesh Generation. In: Handbook of Discrete and Computational Geometry, ch. 25, edited by J.E. Goodman and J. O'Rourke, 2nd edn., pp. 563–582. CRC Press LLC, Bocal Raton, FL (2004)
4. Bern, M., Edelsbrunner, H., Eppstein, D., Mitchell, S., Tan, T.S.: Edge Insertion for Optimal Triangulation. Discrete and Computational Geometry 10(1), 47–65 (1993)
5. Bern, M., Eppstein, D.: 'Mesh Generation and Optimal Triangulation. Computing in Euclidean Geometry. World Scientific, Singapore (1992)
6. Chazelle, B.: Triangulating a simple polygon in linear time. Discrete & Computational Geometry 6(5), 485–524 (1991)
7. Chew, P.: Constrained Delaunay triangulation. In: Proc. 3rd ACM Symp. on Computational Geometry, pp. 215–223 (1987)
8. Collins, G.E.: Quantifier Elimination for Real Closed Fields by Cylindrical Algebraic Decomposition. In: Proc. 2nd GI Conf. Automata Theory Formal Lang. Lecture Notes Comput. Sci. 33, pp. 134–183. Springer, Heidelberg (1975)
9. Field, D.A.: Laplacian Smoothing and Delaunay Triangulations. Communications in Applied Numerical Methods 4, 709–712 (1988)

10. Koltun, V.: Almost tight upper bounds for vertical decompositions in four dimensions. J. ACM 51(5), 699–730 (2004)
11. Lee, D.T., Lin, A.K.: Generalized Delaunay triangulation for planar graphs. Discrete & Computational Geometry 1, 201–217 (1986)
12. Matoušek, J., Sharir, M., Welzl, E.: A subexponential bound for linear programming. Algorithmica 16, 498–516 (1996)
13. Sharir, M., Agarwal, P.K.: Davenport-Schinzel Sequences and Their Geometric Applications. Cambridge University Press, Cambridge (1995)
14. Shewchuk, J.R.: Delaunay Refinement Algorithms for Triangular Mesh Generation. Computational Geometry: Theory and Applications 22(1-3), 21–74 (2002)
15. Seidel, R.: Constrained Delaunay triangulations and Voronoi diagrams with obstacles. In: 1978-1988 Ten Years IIG, pp. 178–191 (1988)

New Algorithm for Field Splitting in Radiation Therapy

Xiaodong Wu[1,2], Xin Dou[1], John Bayouth[2], and John Buatti[2]

[1] Department of Electrical and Computer Engineering
[2] Department of Radiation Oncology
University of Iowa, Iowa City, IA 52246, USA

Abstract. In this paper, we study an interesting geometric partition problem, called *optimal field splitting*, which arises in *Intensity-Modulated Radiation Therapy* (IMRT). In current clinical practice, a multi-leaf collimator (MLC) is used to deliver the prescribed intensity maps (IMs). However, the *maximum leaf spread* of an MLC may require to split a large intensity map into several overlapping sub-IMs. We develop the first optimal linear time algorithm for solving the field splitting problem while minimizing the total complexity of the resulting sub-IMs. Meanwhile, our algorithm strives to minimize the maximum beam-on time of those sub-IMs. Our basic idea is to formulate the field splitting problem as computing a shortest path in a directed acyclic graph, with a special "layered" structure. The edge weights of the graph satisfy the Monge property, which enables us to speed up the algorithm to optimal linear time. To minimize the maximum beam-on time of the resulting sub-IMs, we consider an interesting min-max slope path problem in a monotone polygon which is solvable in linear time. The min-max slope path problem is of its own interest.

1 Introduction

In this paper, we study an interesting geometric partition problem, called *optimal field splitting with feathering*, which arises in *Intensity-Modulated Radiation Therapy* (IMRT). IMRT is a modern cancer therapy technique that aims to deliver a highly conformal radiation dose to a target tumor while sparing the surrounding normal tissues. The quality of IMRT crucially depends on the ability to accurately and efficiently deliver the prescribed dose distributions of radiation, commonly called *intensity maps* (IMs). An intensity map is specified by a set of nonnegative integers on a 2-D grid. The number in a grid cell indicates the amount of radiation to be delivered to the corresponding body region.

An advanced tool today for IM delivery is the multileaf collimator (MLC) [9] (Figure 1 (a)). An MLC consists of a number of pairs of tungsten alloy leaves of the same rectangular shape and size. The leaves can move left and right to form a rectilinear region, called an *MLC-aperture*. The cross-section of a cylindrical radiation beam is shaped by an MLC-aperture. Each MLC-aperture is associated with an integer representing the radiation units delivered by its radiation beam.

T. Tokuyama (Ed.): ISAAC 2007, LNCS 4835, pp. 692–703, 2007.

The mechanical design of the MLCs restricts what kinds of beam-shaping regions are allowed [9]. One common constraint is called the *maximum leaf spread*: Each MLC leaf can only travel away from the vertical center line of the MLC within a certain threshold distance. Note that during the delivery of an IM, the vertical center line of the MLC is always aligned with the center of the IM. Geometrically, the maximum leaf spread means the rectilinear y-monotone polygon corresponding to each MLC-aperture has a maximum horizontal "width" $\leq w$ (e.g., $w = 14.5cm$ for the Varian MLCs).

One popular IMRT approach for delivering IMs using an MLC is the "step-and-shoot" technique [9]. Mathematically, the "step-and-shoot" delivery planning can be viewed as the following IM segmentation problem: Given an intensity map A defined on a 2-D $m \times n$ grid, decompose A into the form of $A = \sum_{k=1}^{\kappa} \alpha_k S_k$, where S_k is a special 0-1 matrix specifying an MLC-aperture, α_k is the amount of radiation delivered through S_k, and κ is the number of MLC-apertures used to deliver A . To deliver the IM, MLC leaves move to form each of those κ MLC-apertures, S_i, and to deliver α_i units of radiation. The reader is referred to [9] for more details on the step-and-shoot IMRT technique.

Two criteria are usually used to measure the efficiency of the step-and-shoot delivery: (1) the *beam-on time* which is given by $\sum_{i=1}^{\kappa} \alpha_i$, and (2) the number κ of MLC-apertures used. The beam-on time is the actual time that the patient is exposed under the radiation beams. Increasing the efficiency of the delivery is crucial since as the total treatment delivery time increases, there are several bad clinical consequences: patient motion increases and tumor cells may be able to better repair themselves during the treatment. All these will make the treatment less effective. The beam-on time and the number of MLC-apertures are closely associated with the complexity of the IM A, which, however, is not well defined. In this paper, we use the *sum of positive gradients* of the intensity map along the direction of leaf motion to measure the complexity $C(A)$ of an IM A [2, 8, 4], more precisely, $C(A) = \sum_{i=1}^{m} \left(a_{i,1} + \sum_{j=2}^{n} \max(0, a_{i,j} - a_{i,j-1}) \right)$. That complexity measure has been used in medical physics literature. For example, Bortfeld *et al* adopted the same complexity measure of an IM to quantify the tradeoff between complexity and conformality of an IMRT plan [2]. Siochi also used that IM complexity to measure the delivery quality in a method for increasing the spatial resolution of the MLC leaf width without using additional hardware [8].

In current clinical radiation therapy, large intensity maps frequently occur [10]. Due to the maximum leaf spread constraint of the MLC design, a large IM needs to be split into several sub-IMs each being delivered separately using the step-and-shoot delivery technique. However, such splitting may result in prolonged beam-on time and increased complexity, and thus compromise the treatment quality. The *field splitting problem*, roughly speaking, is to split an IM of a large size into multiple sub-IMs whose sizes are no larger than a threshold size, such that the treatment quality is optimized.

One simple way to split a large IM is to use straight lines, yielding *abutting* sub-IMs. One of the problems associated with this method is the field mismatching problem that occurs in the field junction region due to the uncertainties in

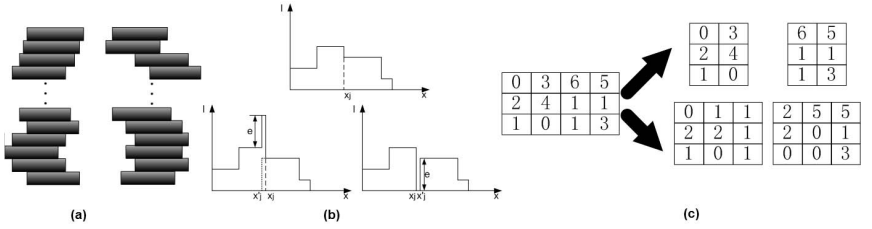


Fig. 1. Illustration of (a) multileaf collimator (b) hotspot and coldspot caused by field splitting, the top profile shows the desired profile split at x_j, due to field mismaching, the left end of right field is positioned at x'_j and the fields may overlap as in bottom left to cause hotspot or be separated as in bottom right to cause coldspot, and (c) field splitting without (top) and with (bottom) feathering. The feathering region consists of two columns.

setup and organ motion [7, 10]. If the borders of two abutting sub-IMs do not precisely align each other, it may result in *hotspots* or *coldspots*, as illustrated in Figure 1(b). To alleviate the field mismatching problem, a commonly used medical practice is to apply a so-called *field feathering* technique [7, 10]. Using this technique, a large IM A is split into a set of sub-IMs, subjected to the maximum leaf spread constraint, and any two adjacent sub-IMs overlap over a central *feathering region*. Note that in the latter method, each cell of the feathering region can belong to two adjacent sub-IMs, with non-negative intensity value in both sub-IMs, as illustrated in Figure 1 (c). While splitting an IM into multiple sub-IMs with minimum total complexity, it is also desirable to minimize the maximum beam-on time of the resulting sub-IMs. The motivation for this optimization is that, during the delivery of each sub-IM, the patient may move, and the longer the beam-on time of a sub-IM, the higher the chance of body motion is.

A few field splitting algorithms have been recently reported in the literature. To our best knowledge, Kamath *et al.* [6] first gave an $O(mn^2)$ time algorithm to split an $m \times n$ IM using vertical lines into *at most three* sub-IMs while minimizing the total beam-on time. Wu [11] formulated the field splitting problem for an arbitrary field width using vertical lines as a k-link shortest path problem and developed an $O(mnw)$ time algorithm, where w is the maximum allowed field width. Kamath *et al.* recently studied the field splitting with feathering [7]. However, their algorithm is optimal only for the case that the input IM has one row and the width of the IM is $\leq 3w$. Chen and Wang [3] further developed an $O(mn+m\xi^{d-2})$ time algorithm for optimally splitting an IM of size $m \times n$, where d is the number of the resulting sub-IMs and ξ is the reminder of n divided by w. Very recently, field splitting while addressing delivery accuracy was studied in [12, 4]. Although it is useful to consider field splitting with feathering to minimize the total complexity, to our best knowledge, no such algorithm has been studied.

In this paper, we study the following **optimal field splitting with balanced beam-on times (OFSB)** problem. Given an IM $A = (a_{i,j})_{m \times n}$ of size $m \times n$,

an integral maximum leaf spread $w > 0$, and the width range $[\delta \,..\, \Delta]$ of each feathering region ($0 < \delta < \Delta < w$), split A into a sequence of $d = \lceil \frac{n-\delta}{w-\delta} \rceil$ (≥ 2) sub-IMs, such that: (1) the width of each sub-IM is w; (2) any two neighboring sub-IMs in the sequence overlap each other and the width of the overlapping (feathering) region ranges from δ to Δ; (3) no sub-IM overlaps completely with its neighboring sub-IM(s); and (4) the total sum of the complexity of all these d sub-IMs is minimized. The resulting d sub-IMs, however, may have a large minimum beam-on time. We thus seek to further decompose the induced (d-1) feathering regions of those d sub-IMs, yielding a set $\mathcal{S}$ of d sub-IMs $\{S_1, S_2, \ldots, S_d\}$ (from left to right), such that the maximum $M_{bot}(\mathcal{S})$ of all the minimum beam-on times $T_{bot}(\cdot)$ of these sub-IMs in $\mathcal{S}$ is minimized, while imposing the lower bound on the total complexity of the splitting $\mathcal{S}$. Note that d is the minimum number of sub-IMs needed. We may use more sub-IMs, which, however, could undesirably increase the total treatment time. We also assume that each sub-IM has a maximum width w since we can introduce columns filled with 0's to the sub-IM without increasing its complexity.

We present the first linear time algorithm for solving the above field splitting problem. In our algorithm, we first model the computation of an "optimal" set of (d-1) feathering regions (i.e., with minimum total increase of the complexity) as a shortest path problem in a directed acyclic graph (DAG) with $O(n)$ vertices and $O(n(\Delta - \delta))$ edges. This DAG has a special "layered" structure, which consists of d layers of vertices with any two adjacent layers inducing a bipartite graph. We are able to calculate each edge weight in constant time after a certain preprocessing. Moreover, the edge weights satisfy the Monge property [1] which enables us to speed up our algorithm to optimal $O(mn)$ time. Then, the decomposition of the resulting feathering regions is modeled as computing a min-max slope path problem in a special monotone polygon. We develop an interesting geometric algorithm, which runs in linear time, for solving the min-max slope path problem. Due to the page limit, all proofs have been omitted.

2 Computing an Optimal Set of Feathering Regions

In this section, we compute in an optimal $O(mn)$ time a set $\mathcal{F}$ of (d-1) feathering regions for a given instance of the OFSB problem, such that the decomposition of the feathering regions in $\mathcal{F}$ yields a splitting of the input IM A with minimum total complexity. The problem is modeled as computing a shortest path in a DAG, for which we can exploit the Monge property to speed up the computation.

2.1 The Shortest Path Model

Denote by $A[j]$ the j-th column of IM A, and $A[j \,..\, k]$ consists of all rows of A from Column j to Column k. Since the width of each sub-IM is fixed as w, d vertical lines $\{j_1, j_2, \ldots, j_d\}$ are needed to determine the starting column of

each sub-IM in the splitting (including the first vertical line which is always corresponding to the first column of A, i.e., $j_1 = 1$). The k-th feathering region F_k consists of multiple columns of A starting from Column j_{k+1} to Column $j_k + w - 1$ denoted by $A[j_k - \delta + 1 \,..\, j_k]$. $F_k = (f_{i,j})$ is somehow decomposed into $F_k^{(0)} = (f_{i,j}^{(0)})$ and $F_k^{(1)} = (f_{i,j}^{(1)})$ such that $F_k = F_k^{(0)} + F_k^{(1)}$ (i.e., the value of every element in F_k is decomposed into two non-negative integers, one in $F_k^{(0)}$ and the other in $F_k^{(1)}$). Then, a feasible splitting $\mathcal{S} = \{S_1, S_2, \ldots, S_d\}$ of A is, as follows. For each $k = 1, 2, \ldots, d$, $S_k = F_{k-1}^{(1)} \,||\, A[j_{k-1} + w \,..\, j_{k+1} - 1] \,||\, F_k^{(0)}$, where $||$ is a concatenation operator, $F_0^{(1)} = F_d^{(0)} = \varnothing$, and $j_{d+1} = n + 1$. The decomposition of each feathering region F_k may increase the total complexity. Our goal is to find a set $\mathcal{F}$ of (d-1) feathering regions such that the total increase of the complexity resulting from the decomposition is minimized. We next model this problem as computing a shortest path in a weighted directed acyclic graph $G = (V, E)$.

1) The graph G has d layers of vertices, where each vertex in the k-th layer (denoted by L_k) defines a possible starting column of the k-th sub-IM. The first layer L_1 consists of only one vertex v_1. For the k-th sub-IM S_k, there are $k - 1$ (resp., $d - k$) sub-IMs to the left (resp., right) of it. Thus, the rightmost (resp., leftmost) possible starting column of sub-IM S_k is $(k-1)(w-\delta)+1$ (resp., $(k-1)(w-\delta)+1-\mu$), where $\mu = (n-\delta) \bmod (w-\delta)$. Thus, each layer L_k ($k = 2, \ldots, d$) contains μ vertices $\{v_j \mid (k-1)(w-\delta)+1-\mu \le j \le (k-1)(w-\delta)+1\}$. Note that no sub-IM overlaps with non-neighboring sub-IMs to achieve the minimum number of sub-IMs. Hence, the maximum width Δ of a feathering region is less than $(w - \delta)$, and the layers are mutually exclusive.

2) For each vertex $v_j \in L_k$ in G, there is a directed edge from v_j to every $v_{j'} \in L_{k+1}$ as long as the two corresponding sub-IMs overlap each other with an overlapping region of width between δ and Δ. Thus, each edge $(v_j, v_{j'})$ defines a feathering region $A[j' \,..\, j + w - 1]$.

3) For any edge $e = (v_j, v_{j'})$ in G, we compute the minimum increased complexity of the corresponding feathering region, which is precisely defined in Section 2.2, and assign it as the weight of the edge, denoted by $c(v_j, v_{j'})$.

4) For the ease of our algorithm description, we introduce a dummy vertex t. Each vertex in the d-th layer L_d has a directed edge to t with a weight of 0.

Our algorithm then computes a shortest v_1-to-t path $v_1 \to v_{j_2} \to \cdots \to v_{j_{d-1}} \to v_{j_d} \to t$ in G, where $v_{j_k} \in L_k$. Obviously, the d sub-IMs defined by the starting columns in $\{1, j_2, \ldots, j_{d-1}, j_d\}$ specifies a feasible splitting $\mathcal{S}^*$ of A. The total increase of the complexity due to the splitting of $\mathcal{S}^*$ equals the total path weight $c(p)$, which is minimized. Thus, $\mathcal{S}^*$ is a splitting of A minimizing the total increase of the complexity, i.e., the total complexity of $\mathcal{S}^*$ is minimized.

We next show how to efficiently compute the minimum increase of the complexity for all feathering regions in $O(m)$ time after an $O(mn)$ time preprocessing. In Section 2.3, we exploit the Monge property of the graph G to speed up the shortest path computation, to optimal linear $O(mn)$ time.

2.2 Computing the Minimum Increase of the Complexity for a Feathering Region

In this section, we characterize the complexity increase due to the feathering region decomposition. Then, a linear time algorithm is developed for the optimal decomposition of each feathering region.

Characterizing the Increase of the Complexity. Consider a feathering region $F_k = A[l\,..\,r]$, which is the overlapping region of sub-IMs S_{k-1} and S_k with $\delta \leq r - l = W \leq \Delta$ $(1 < l < r < n)$, where W is the region width. Assume that the decomposition of F_k is $F_k^{(0)} = (x_{i,j})_{m\times W}$ plus $F_k^{(1)} = (y_{i,j})_{m\times W}$. Defining $R(A[l\,..\,r])$ as $\sum_{i=1}^{m}\sum_{j=l}^{r+1}\max\{0, a_{i,j} - a_{i,j-1}\}$, it is mathematically trivial to show the *increase* of the complexity due to the decomposition is $R(F_k^{(0)}) + R(F_k^{(1)}) - R(F_k)$. We next develop a linear time algorithm for optimal decomposition of a feathering region, minimizing the increased complexity.

Observing that the decomposition of F_k can be performed row by row, we define the following **optimal vector decomposition (OVD)** problem. Define the *weight* $W_{ovd}(\mathbf{z})$ of a vector $\mathbf{z} = (z_1, z_2, \ldots, z_N)$ as $\sum_{j=2}^{N}\max\{0, z_j - z_{j-1}\}$. Given a non-negative integer vector $\mathbf{b} = (b_1, b_2, \ldots, b_N)$, decompose $\mathbf{b}$ into two non-negative vectors $\mathbf{x} = (x_1, x_2, \ldots, x_N)$ and $\mathbf{y} = (y_1, y_2, \ldots, y_N)$, such that (1) $x_1 = b_1$ and $y_N = b_N$; (2) for each $j = 1, 2, \ldots, N$, $b_j = x_j + y_j$, and (3) the total weight of $\mathbf{x}$ and $\mathbf{y}$ (i.e., $W_{ovd}(\mathbf{x}) + W_{ovd}(\mathbf{y})$) is minimized.

Then, for a given feathering region $F_k = A[l\,..\,r]$, we may view each *extended* row $(a_{i,l-1}, a_{i,l}, \ldots, a_{i,r}, a_{i,r+1})$ as a vector $\mathbf{a}_i$. Applying the OVD algorithm, $\mathbf{a}_i$ is decomposed into two vectors, $\mathbf{x}_i = (a_{i,l-1}, x_{i,l}, \ldots, x_{i,r}, 0)$ and $\mathbf{y}_i = (0, y_{i,l}, \ldots, y_{i,r}, a_{i,r+1})$, with $W_{ovd}(\mathbf{x}_i) + W_{ovd}(\mathbf{y}_i)$ being minimized. Clearly, $(\mathbf{x}_i)_{i=1}^m$ and $(\mathbf{y}_i)_{i=1}^m$ can be used to specify an optimal decomposition of F_k.

Chen and Wang [3] also studied this OVD problem for a different purpose and developed a linear time algorithm. We independently solved the OVD problem in linear time. Our algorithm is ready to be extended to more general cases.

Linear Time Algorithm for Optimal Vector Decomposition (OVD) Problem. This section presents our optimal $O(N)$ time algorithm for computing an optimal decomposition of a given vector $\mathbf{b} = (b_1, b_2, \ldots, b_N)$. The OVD problem is modeled as computing a shortest path in a directed acyclic graph (DAG) of pseudo-polynomial size. By exploiting the convexity of the edge weight functions of the graph, we show that the OVD problem can be optimally solved in linear time without explicitly constructing and searching the whole graph. The solution space of the optimal decomposition of a vector $\mathbf{b}$ is then judiciously characterized, which is crucial for our OFSB algorithm.

Define a non-negative vector $\mathbf{x}^* = (x_1, x_2, \ldots, x_N)$ as follows.

$$x_j = \begin{cases} b_1, & j = 1 \\ \max\{0, x_{j-1} - \max\{0, b_{j-1} - b_j\}\}, & 1 < j < N \\ 0, & j = N \end{cases} \tag{1}$$

Then, for every $j = 1, 2, \ldots, N$, $y_j = b_j - x_j$. Obviously, $\mathbf{x}^*$ and $\mathbf{y}^*$ is a feasible decomposition of $\mathbf{b}$. We can further prove that the total weight of $\mathbf{x}^*$ and $\mathbf{y}^*$is minimized. Thus, we have Lemma 1.

Lemma 1. *Given a non-negative vector* $\mathbf{b} = (b_1, b_2, \ldots, b_N)$*, an optimal decomposition* $\mathbf{x}^*$ *and* $\mathbf{y}^*$ *of* $\mathbf{b}$ *can be computed in* $O(N)$ *time; furthermore,* $W_{ovd}(\mathbf{x}^*) = 0$ *and the total weight of* $\mathbf{x}^*$ *and* $\mathbf{y}^*$*,* $W_{ovd}(\mathbf{x}^*) + W_{ovd}(\mathbf{y}^*) = \max\{b_N, \sum_{j=2}^{N} \max\{0, b_j - b_{j-1}\}\}$

Lemma 2 further characterizes the optimal solution space of the OVD problem. Denote by $\rho(\mathbf{b})$ the total weight of an optimal decomposition of $\mathbf{b}$., i.e., $\rho(\mathbf{b}) = W_{ovd}(\mathbf{x}^*) + W_{ovd}(\mathbf{y}^*)$.

Lemma 2. *Given a non-negative vector* $\mathbf{b} = (b_1, b_2, \ldots, b_N)$ *and a nonnegative integer* τ*, there exists an optimal decomposition* $\mathbf{x}$ *and* $\mathbf{y}$ *of* $\mathbf{b}$ *with* $W_{ovd}(\mathbf{x}) = \tau$ *if and only if* $\tau \leq \rho(\mathbf{b}) - b_N$*. The decomposition can be computed in* $O(N)$ *time.*

Computing the Minimum Increase of the Complexity. Based on Lemma 1, the minimum increase of the complexity for F_k, denoted by $\mathcal{I}_{cpl}(F_k)$, is $\sum_{i=1}^{m} \max\{0, a_{i,r+1} - \sum_{j=l}^{r+1} \max\{0, a_{i,j} - a_{i,j-1}\}\}$. We thus can introduce an additional matrix $B = (b_{i,j})_{m \times n}$ such that $b_{i,j} = \sum_{k=1}^{j+1} \max\{0, a_{i,k} - a_{i,k-1}\}\}$ ($a_{i,0} = a_{i,n+1} = 0$). The matrix B can be computed in $O(mn)$ time. Then, $\mathcal{I}_{cpl}(F_k)$ for any feathering region F_k can be obtained in $O(m)$ time.

Lemma 3. *After an* $O(mn)$ *time preprocessing, the minimum increase* $\mathcal{I}_{cpl}(F_k)$ *of the complexity for any feathering region* $F_k = A[l\,..\,r]$ *can be computed in* $O(m)$ *time, with* $\mathcal{I}_{cpl}(F_k) = \sum_{i=1}^{m} \max\{0, a_{i,r+1} - \sum_{j=l}^{r+1} \max\{0, a_{i,j} - a_{i,j-1}\}\}$*, and the decomposition of* F_k *can be obtained in* $O(mn)$ *time.*

2.3 Speeding Up the Computation of the Shortest v_1-to-t Path in G

In this section, we exploit the Monge property [1] of the graph $G = (V, E)$ .This enables us to compute a shortest v_1-to-t path in G in $O(n) = O(|V|)$ time.

Lemma 4. *Given four vertices* $v_{j'}, v_{j'+1} \in L_k$ *and* $v_{j''}, v_{j''+1} \in L_{k+1}$ *in* G *with* $2 \leq k < d$*,* $c(v_{j'}, v_{j''}) + c(v_{j'+1}, v_{j''+1}) \leq c(v_{j'}, v_{j''+1}) + c(v_{j'+1}, v_{j''})$.

The Monge property as shown in Lemma 4 can be used to speed up the computation of the shortest v_1-to-t path in G. For every vertex v_l in the k-th layer L_k, let $sw_k(l)$ denote the shortest path length from v_1 to $v_l \in L_k$ in G. Clearly, $sw_k(l) = \min\{sw_{k-1}(l') + c(v_{l'}, v_l) \mid v_{l'} \in L_{k-1}$ and $w - \Delta \leq l - l' \leq w - \delta\}$. Recall that an edge $(v_{l'}, v_l) \in E$ if and only if $v_{l'} \in L_{k-1}$, $v_l \in L_k$, and $w - \Delta \leq l - l' \leq w - \delta$. Hence, the set of all outgoing edges of each vertex $v_{l'}$ and the set of all incoming edges of each v_l can be represented implicitly (such that we can access any edge of G in $O(1)$ time and compute its weight in $O(m)$ time as shown in Section 2.2). Note that the Monge property is normally defined on a matrix [1]. Lemma 4 actually shows the Monge property of the

matrix containing the path weight $sw_{k-1}(l') + c(v_{l'}, v_l)$ for every edge $(v_{l'}, v_l)$ between the vertices on two consecutive layers L_k and L_{k+1} of G, with $1 < k < d$. Thus, applying the matrix-searching technique in [1], it takes $O(m(w-\delta))$ time to compute all shortest paths from v_1 to all vertices on the k-th layer when knowing all $sw_{k-1}(l')$'s of Layer L_{k-1}. Hence, a shortest v_1-to-t path in G can be obtained in $O(dm(w-\delta)) = O(mn)$ time.

Lemma 5. *Given an instance of the OFSB problem, an optimal set of feathering regions can be computed in $O(mn)$ time.*

After obtaining an optimal set of feathering regions, we can decompose each extended row of every feathering region by Lemma 1, yielding an optimal splitting. However, the resulting sub-IMs may have an undesirably large minimum beam-on time. We thus consider the following **min-max beam-on time (MMBoT)** problem: Given an optimal set of feathering regions $\{F_k \mid k = 1, 2, \ldots, d-1\}$, decompose the feathering regions to achieving a set $\mathcal{S}$ of d sub-IMs $\{S_1, S_2, \ldots, S_d\}$, such that the maximum $M_{bot}(\mathcal{S})$ of all the minimum beam-on times $T_{bot}(\cdot)$ of these sub-IMs in $\mathcal{S}$ (i.e., $M_{bot}(\mathcal{S}) = \max_{S_k \in \mathcal{S}} T_{bot}(S_k)$) is is minimized, while imposing the lower bound on total complexity of the splitting.

2.4 Linear Time Algorithm for Solving the MMBoT Problem

In this section, we formulate the MMBoT problem as computing a polygonal path in a polygon, such that the maximum slope of the segments on the path is minimized, which can be solved in linear time.

The Min-max Slope Path Model. Assume that each feathering region $F_k = A[j_{k+1} .. j_k + w - 1]$ is decomposed into $F_k^{(0)}$ and $F_k^{(1)}$, and denote by $\alpha_k(i)$ and $\beta_k(i)$ the weights of each row i of $F_k^{(0)}$ and $F_k^{(1)}$, respectively, as defined in Section 2.2. Since we impose the lower bound on the total complexity of the splitting while performing the decomposition of F_k, by Lemma 1, $\alpha_k(i) + \beta_k(i)$ is a fixed constant, denoted by $\rho_k(i)$, and $0 \le \alpha_k(i), \beta_k(i) \le \rho_k(i)$. Then, for each $k = 1, 2, \ldots, d$, as shown in Section 2.1, the k-th sub-IM in the splitting $\mathcal{S}$, $S_k = F_{k-1}^{(1)} \,||\, A[j_{k-1} + w .. j_{k+1} - 1] \,||\, F_k^{(0)}$. For a given set of $(d-1)$ feathering regions, $A[j_{k-1} + w .. j_{k+1} - 1]$ in each S_k is fixed, and thus we let $c_k(i) = \sum_{l=j_{k-1}+w}^{j_{k+1}-1} \max\{0, a_{i,l} - a_{i,l-1}\}$, which is a constant. Note that the minimum beam-on time $T_{bot}(B)$ of an IM $B = (b_{i,j})_{m' \times n'}$ equals to $\max_{i=1}^{m'}\{b_{i,1} + \sum_{j=2}^{n'} \max\{0, b_{i,j} - b_{i,j-1}\}\}$ [5]. Hence, $T_{bot}(S_k) = \max_{i=1}^{m}(\beta_{k-1}(i) + c_k(i) + \alpha_k(i))$, where $\alpha_k(i)$ and $\beta_k(i)$ are varying depending on the decompositions of F_k's. Then, the maximum $M_{bot}(\mathcal{S})$ of all the minimum beam-on times of the sub-IMs in $\mathcal{S}$ is $\max_{S_k \in \mathcal{S}} \max_{i=1}^{m}(\beta_{k-1}(i) + c_k(i) + \alpha_k(i))$, that is, $M_{bot}(\mathcal{S}) = \max_{i=1}^{m} \max_{k=1}^{d}(\beta_{k-1}(i) + c_k(i) + \alpha_k(i))$, where $\beta_0(i) = \alpha_d(i) = 0$.

Thus, to minimize $M_{bot}(\mathcal{S})$, we need to solve the following problem: Given a vector $\rho = \{\rho_0, \rho_1, \ldots, \rho_{d-1}, \rho_d\}$ with $\rho_0 = \rho_d = 0$ and $\rho_k \ge 0$ $(k = 1, 2, \ldots, d-1)$, and a vector $c = (c_1, c_2, \ldots, c_d)$ with $c_k \ge 0$ for every $k = 1, 2, \ldots, d$, decompose ρ into two vectors $\alpha = (\alpha_0, \alpha_1, \ldots, \alpha_d)$ and $\beta = (\beta_0, \beta_1, \ldots, \beta_d)$, such

that: (1) $\alpha_k + \beta_k = \rho_k$ for each $k = 0, 1, \ldots, d$; (2) $0 \leq \alpha_k, \beta_k \leq \rho_k$; and (3) $\max_{k=1}^{d}(\beta_{k-1} + c_k + \alpha_k)$ is minimized. Without loss of generality, we assume that $\rho_k > 0$ for $k = 1, 2, \ldots, d-1$. Interestingly, we can model this problem as a **min-max slope path (MMSP)** problem in a polygon, as follows.

Define a monotone polygon $\mathcal{P}$ in the 2-D **x-y** plane, whose boundary consists of two **y**-monotone polygonal chains, the upper chain $\mathcal{C}_u$ and the lower one $\mathcal{C}_l$. Both $\mathcal{C}_u$ and $\mathcal{C}_l$, each consisting of $d+1$ chain vertices, start at the point $s(0, 0)$ and end at the point $t(d, \sum_{i=1}^{d} c_i + \sum_{i=0}^{d} \rho_i)$. The k-th vertex on the lower (resp., upper) chain $\mathcal{C}_l$ (resp., $\mathcal{C}_u$) is at $\underline{B}_k(k, \sum_{i=1}^{k} c_i + \sum_{i=0}^{k-1} \rho_i)$ (resp., $\overline{B}_k(k, \sum_{i=1}^{k} c_i + \sum_{i=0}^{k} \rho_i)$) for $k = 1, 2, \ldots, d-1$. Let $x(P)$ (resp, $y(P)$) denote the x-coordinate (resp., y-coordinate) of a point P. The *min-max slope path* problem seeks a polygonal path $L = P_0 P_1 \ldots P_{q-1} P_q$ in $\mathcal{P}$ such that (1) $P_0 = s$ and $P_q = t$ and (2) the maximum slope of the line segments on L, denoted by $MS(L)$ (i.e., $MS(L) = \max_{i=1}^{q} \frac{y(P_i) - y(P_{i-1})}{x(P_i) - x(P_{i-1})}$), is minimized. Such a path L is called a min-max slope path from s to t.

For any feasible decomposition of ρ, $\alpha = (\alpha_0, \alpha_1, \ldots, \alpha_{d-1}, \alpha_d)$ and $\beta = (\beta_0, \beta_1, \ldots, \beta_{d-1}, \beta_d)$, we define a polygonal path $L = P_0 P_1, \ldots, P_{d-1} P_d$ with $P_0 = s$, $P_d = t$, and $P_k = (k, y(\underline{B}_k) + \alpha_k)$ for $k = 1, 2, \ldots, d-1$. It is not difficult to see that L is in the polygon $\mathcal{P}$, and $MS(L) = \max_{k=1}^{d}(\beta_{k-1} + c_k + \alpha_k)$.

Computing a Min-Max Slope *s*-to-*t* Path. In the polygon $\mathcal{P}$, the line segments $(\underline{B}_k, \overline{B}_k)$ (resp., $(\overline{B}_k, \underline{B}_{k+1})$) are called *type I diagonals* (resp., *type II diagonals*) of $\mathcal{P}$. Let $\overline{v}_k$ and $\underline{v}_k$ be the two end points of a diagonal d_k with $\overline{v}_k$ having a larger y-coordinate than $\underline{v}_k$. Denote by $D(s, \overline{v}_k)$ and $D(s, \underline{v}_k)$, respectively, the min-max slope paths from s to $\overline{v}_k$ and $\underline{v}_k$. There is a unique common vertex u of both $D(s, \overline{v}_k)$ and $D(s, \underline{v}_k)$, which is the furthest one from s on either path. Obviously $D(u, \overline{v}_k)$ and $D(u, \underline{v}_k)$ have no common edge. Denote by D_k the region enclosed by $D(u, \overline{v}_k)$, $D(u, \underline{v}_k)$, and d_k. The vertex u is called the *tip point* of D_k. Lemma 6 characterizes the inward-convexity of $D(u, \overline{v}_k)$ and $D(u, \underline{v}_k)$. Lemma 7 shows that a min-max slope s-to-t path in $\mathcal{P}$ may only use the polygonal vertices of $\mathcal{P}$.

Lemma 6. *If neither of $D(u, \overline{v}_k)$ and $D(u, \underline{v}_k)$ is empty, both $D(u, \overline{v}_k)$ and $D(u, \underline{v}_k)$ are inward-convex polygonal chains.*

Lemma 7. *There exists a min-max slope s-to-t path $L^* = P_0 P_1 \ldots P_{d-1} P_d$ such that $P_0 = s$, $P_d = t$, and P_k is a polygonal vertex of $\mathcal{P}$ for every $k = 1, 2, \ldots, d-1$.*

Now, the following algorithm can find a min-max slope s-to-t path L^* in $\mathcal{P}$:

Step 1: Construct D_1 by connecting s to $\overline{v}_1 = \overline{B}_1$ and $\underline{v}_1 = \underline{B}_1$. Set $k = 1$ and $d_k = \overline{\overline{v}_k \underline{v}_k}$.

Step 2: Let u be the tip point of D_k, at which the two subchains $\overline{u_a u_{a+1} \ldots u_b}$ and $\overline{u_a u_{a-1} \ldots u_0}$ diverge, where $u = u_a$, $\overline{v}_k = u_0$, $\underline{v}_k = u_b$. Consider the next diagonal $d_{k+1} = \overline{\overline{v}_{k+1} \underline{v}_{k+1}}$, where $\underline{v}_{k+1} = \overline{v}_k$. Based on the definition of a diagonal, $\overline{v}_{k+1}$ is also determined. Starting from u_0, scan the sequence $u_0, u_1, \ldots, u_b$ and let j be the smallest index for which $\overline{\overline{v}_{k+1} u_j}$ becomes a *supporting* segment

(a line segment is a supporting segment of an open convex curve if it has at least one point on the convex curve and the whole convex curve is on one side of the line segment) of the boundary of D_k. Here we have four possible cases:

(i) d_{k+1} is a type I diagonal and $j \leq a$ (Figure 2(a)). Remove all edges $\overline{u_i u_{i+1}}$ for $0 \leq i \leq j-1$ and add edge $\overline{u_j \overline{v}_{k+1}}$.
(ii) d_{k+1} is a type I diagonal and $j > a$ (Figure 2(b)). Remove all edges $\overline{u_i u_{i+1}}$ for $0 \leq i \leq j-1$ and add $\overline{u_j \overline{v}_{k+1}}$; u_j becomes the tip point of D_{k+1}.
(iii) d_{k+1} is a type II diagonal and $j \leq a$ (Figure 2(c)). Remove all edges $\overline{u_i u_{i+1}}$ for $j \leq i \leq b-1$ and add edge $\overline{u_j \overline{v}_{k+1}}$; u_j becomes the tip point of D_{k+1}.
(iv) d_{k+1} is a type II diagonal and $j > a$ (Figure 2(d)). Remove all edges $\overline{u_i u_{i+1}}$ for $j \leq i \leq b-1$ and add edge $\overline{u_j \overline{v}_{k+1}}$.

Step 3: Let $k = k+1$ and repeat Step 2 until t is reached.

Lemma 8. *A min-max slope s-to-t path in $\mathcal{P}$ can be computed in $O(N)$ time, where N is the number of vertices on $\mathcal{P}$.*

Solution Integerization. The min-max slope s-to-t path L^* in $\mathcal{P}$ computed in Section 2.4 may not intersect with each line $x = k$ ($k = 1, 2, \ldots, d-1$) at a point whose y-coordinate is an integer. This prevents us from defining a valid decomposition of ρ for solving the MMBoT problem.

However, if the maximum slope of a min-max slope s-to-t path L^* in $\mathcal{P}$ is S_{max}, an s-to-t path L'^* in $\mathcal{P}$, which intersects with each line $x = k$ ($k = 1, 2, \ldots, d-1$) at an integer point, can be found by using the following integerization operations and the maximum slope of the segments on L'^* is $\lceil S_{max} \rceil$. Obviously, this is an integer solution with maximum slope of $\lceil S_{max} \rceil$, which is obviously an optimal integer solution.

Suppose $P_k = (x, y)$ and $P_{k+1} = (x + \Delta x, y + \Delta y)$ are two adjacent polygonal vertices on the path L^*. We define the following integerization operation: For each $i = 1, 2, \ldots, \Delta x - 1$, we insert a new point $(x + i, \lceil y + i\Delta y/\Delta x \rceil)$. It is easy to show that none of the slopes of all these segments is larger than $\lceil \Delta y/\Delta x \rceil$.

Performing this integerization operation on each line segment of L^*, we then obtain an s-to-t path L'^* whose vertices are integer points and maximum slope is $\lceil S_{max} \rceil$. Consider each point $P(k, y(\underline{B}_k) + \alpha_k) \in L'^*$ for $k = 1, 2, \ldots, d-1$. Then, ρ_k can be decomposed into α_k and β_k with $\beta_k = \rho_k - \alpha_k$. Thus, ρ can be decomposed into α and β such that $\max_{k+1}^{d}(\beta_{k-1} + c_k + \alpha_k)$ is minimized.

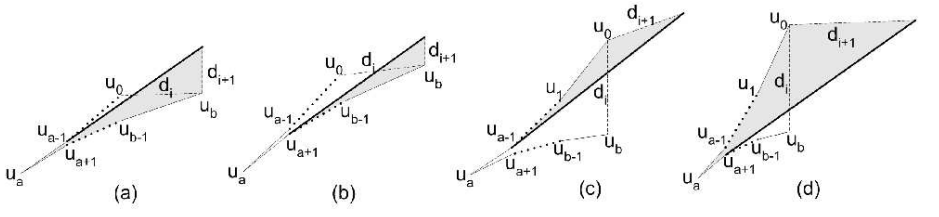


Fig. 2. Illustrating the algorithm for computing the min-max slope path in $\mathcal{P}$. (a), (b), (c), and (d) are four possible cases in Step 2 of the algorithm.

To compute an optimal splitting of IM A, we perform the following steps: (1) Compute an optimal set of $d-1$ feathering regions F_k's. (2) For each row i of these feathering regions, compute $\rho(i) = (\rho_1(i), \rho_2(i), \ldots, \rho_{d-1}(i))$ by Lemma 1. (3) Using our MMSP algorithm to decompose each $\rho(i)$ into $\alpha(i)$ and $\beta(i)$. (4) Based on Lemma 2, each row i of every feathering region F_k can be optimally decomposed into two vectors $\mathbf{x}_k(i)$ and $\mathbf{y}_k(i)$. We thus obtain an optimal spitting $\mathcal{S} = \{S_1, S_2, \ldots, S_d\}$ from the vectors $\mathbf{x}_k(i)$ and $\mathbf{y}_k(i)$ ($k = 1, 2, \ldots, d-1$ and $i = 1, 2, \ldots, m$). We now have the following theorem.

Theorem 1. *Given an IM $A = (a_{i,j})_{m\times n}$,an integral maximum leaf spread $w > 0$, and the width range $[\delta \,..\, \Delta]$ of each feathering region ($0 < \delta < \Delta < w$), the OFSB problem can be solved in $O(mn)$ time.*

3 Implementation and Experiments

To study the performance of our new OFSB algorithm with respect to clinical applications, we implemented the algorithm using Matlab on Windows XP system. We performed experiments on 1000 randomly generated intensity maps for each problem configurations and also 105 sets of clinical intensity maps obtained from the Department of Radiation Oncology, the University of Iowa. We conducted comparisons with Chen and Wang's FSMP algorithm [3], which devotes to minimize the total beam-on time for field splitting with feathering. Comparisons were also made between results before and after splitting using our OFSB algorithm. Both beam-on time and the number of MLC apertures were used as measures of the results. The minimum feathering width δ was set to 3 for comparisons between before and after splitting, while 0 for comparisons between OFSB and FSMP to make sure that the numbers of resulting sub-IMs were equal.

For the randomly generated IMs, the widths of the tested intensity maps were 25, 35, 45, 60, and 75, to get different number of sub-IMs. The maximum intensity level was set to 100. The average increment of the total beam-on time of our OFSB algorithm was only 6.09% over Chen and Wang's FSMP algorithm. In term of the number of MLC apertures, our algorithm slightly outperformed the FSMP algorithm. Our experiments also revealed the increase of the total beam-on time and the number of MLC-apertures were pretty small. For example, while the IMs needed to be split into 7 sub-IMs, the total beam-on time only increased by less than 20% and the number of MLC-apertures by about 30%.

For clinical data, the widths of the tested intensity maps ranged from 15 to 47. The maximum intensity level of each IM was normalized to 100. The maximum leaf spreads used were 14, 16, and 18. For the tested IMs, the total beam-on time using our OFSB algorithm was only slightly larger than that obtained by Chen and Wang's FSMP algorithm, with an average increase of 6.2%; while the number of MLC apertures output by our algorithm was comparable to that by Chen's algorithm, with an average increase of 0.9%.

Our algorithm runs very fast, comparing to Chen and Wang's FSMP algorithm, for most of the intensity maps, the execution time was within a second, while the FSMP algorithm took hundreds of seconds.

References

[1] Aggarwal, A., Klawe, M.M., Moran, S., Shor, P., Wilber, R.: Geometric Applications of a Matrix-Searching Algorithm. Algorithmica 2, 195–208 (1987)
[2] Bortfeld, T., Craft, D., Trofimov, A., Halabi, T., Kufer, K.-H.: Quantifying the Tradeoff between Complexity and Conformality. In: 7th Annual Meeting and Technical Exhibition of the American Association of Physicists in Medicine (AAPM), Seattle, WA. Medical Physics, vol. 32(6) (2005)
[3] Chen, D.Z., Wang, C.: Field Splitting Problems in Intensity-Modulated Radiation Therapy. In: Asano, T. (ed.) ISAAC 2006. LNCS, vol. 4288, pp. 690–700. Springer, Heidelberg (2006)
[4] Chen, D.Z., Healy, M.A., Wang, C., Wu, X.: A New Field Splitting Algorithm for Intensity-Modulated Radiation Therapy. In: 13th Annual Int. Computing and Combinatorics Conference (COCOON) (July 2007)
[5] Engel, K.: A New Algorithm for Optimal Multileaf Collimator Field Segmentation. Discrete Applied Mathematics 152(1-3), 35–51 (2005)
[6] Kamath, S., Sahni, S., Ranka, S., Li, J., Palta, J.: Optimal Field Splitting for Large Intensity-Modulated Fields. Medical Physics 31(12), 3314–3323 (2004)
[7] Kamath, S., Sahni, S., Li, J., Palta, J., Ranka, S., Generalized, A.: Field Splitting Algorithm for Optimal IMRT Delivery Efficiency, The 47th Annual Meeting and Technical Exhibition of the American Association of Physicists in Medicine (AAPM), 2005. Also, Med. Phys. 32(6) 1890 (2005)
[8] Siochi, A.C.: Virtual Micro-Intensity Modulated Radiation Therapy. Medical Physics 27(11), 2480–2493 (2000)
[9] Webb, S.: Intensity-Modulated Radiation Therapy, Institute of Cancer Research and Royal Marsden NHS Trust (2001)
[10] Wu, Q., Arnfield, M., Tong, S., Wu, Y., Mohan, R.: Dynamic Splitting of Large Intensity-Modulated Fields. Phys. Med. Biol. 45, 1731–1740 (2000)
[11] Wu, X.: Efficient Algorithms for Intensity Map Splitting Problems in Radiation Therapy. In: Wang, L. (ed.) COCOON 2005. LNCS, vol. 3595, pp. 504–513. Springer, Heidelberg (2005)
[12] Wu, X., Dou, X.: Optimal Field Splitting with Feathering in Intensity-Modulated Radiation Therapy. In: Kao, M.-Y., Li, X.-Y. (eds.) AAIM 2007. LNCS, vol. 4508, pp. 327–336. Springer, Heidelberg (2007)

In-Place Algorithm for Image Rotation

Tetsuo Asano[1], Shinnya Bitou[1], Mitsuo Motoki[1], and Nobuaki Usui[2]

[1] School of Information Science, JAIST, Japan
[2] Imaging Engineering Div., Products Group, PFU Limited, Japan

Abstract. This paper presents an algorithm for rotating a subimage in place without using any extra working array. Due to this constraint, we have to overwrite pixel values by interpolated values. Key ideas are *local reliability test* which determines whether interpolation at a pixel is carried out correctly without using interpolated values, and *lazy interpolation* which stores interpolated values in a region which is never used for output images and then fills in interpolated values after safety is guaranteed. It is shown that linear interpolation is always safely implemented. An extension to cubic interpolation is also discussed.

1 Introduction

Demand for scanners is growing toward paper-less society. There are a number of problems to be resolved in the current scanner technology. One of them is to detect the direction of a document scanned, i.e., which side is the top of the document. One way is to use OCR technology to read characters which is now common to scanners. Of course, we want to avoid using OCR since it takes time. Another common problem which we address in this paper is correction of rotated documents. If the document contains only characters, then OCR is definitely a solution. Since it is costly, a geometric algorithm for such correction is required. It consists of two phases. In the first phase we detect rotation angle. Some scanners are equipped with a sensor to detect rotation angle. If no such sensor is available, we could rely on another algorithm called Hough transform [1,2] for finding line components to detect rotation angle. To simplify the discussion, we assume a hardware sensor to detect rotation angle.

Once rotation angle is obtained, the succeeding process is rather easy if sufficient working storage is provided. Suppose input intensity values are stored in a two-dimensional array $a[.,.]$ and another array $b[.,.]$ of the same size is available. Then, at each lattice point (pixel) in the rotated coordinate system we compute an intensity value using appropriate interpolation (linear or cubic) using intensity values around the lattice point (pixel) in the input array and then store the computed interpolation value at the corresponding element in the array $b[\,]$. Finally, we output intensity values stored in the array $b[\,]$. It is quite easy. This method, however, requires too much working storage, which is a serious drawback for devices such as scanners in which saving memory is a serious demand for their built-in softwares and their costs. Is it possible to implement the interpolations without using any extra working storage? This is our question in this paper.

T. Tokuyama (Ed.): ISAAC 2007, LNCS 4835, pp. 704–715, 2007.

We propose an efficient algorithm for correcting rotation of a document without using any extra working storage. A simple way of doing this is to compute an interpolation value at each pixel in the rotated coordinate system and store the computed value somewhere in the input array $a[\,]$ near the point in the original coordinate system. Once we store an interpolation value at some element of the array, the original intensity value is lost and it is replaced by the interpolation value. Thus, if the neighborhood of the pixel in the rotated coordinate system includes interpolated values then the interpolation at that point is not correct or reliable. One of keys is a condition to determine whether interpolation at a given pixel is reliable or not, that is, whether any interpolated value is included in the neighborhood or not. Using the condition, we first classify pixels in the rotated coordinate system into reliable and unreliable ones. In the first phase we compute interpolation at each unreliable pixel and keep the interpolation value in a queue, which consists of array elements outside the rotated subimage. Then, in the second phase we compute interpolation at every pixel (x, y) in the rotated coordinate system and store the computed value at the (x, y)-element in the array. Finally, in the third phase for each unreliable pixel (x, y) we move its interpolation value stored in the queue back to the (x, y)-element in the array.

There are increasing demands for such memory-efficient algorithms. The work in this paper would open a great number of possibilities in such directions in applications to computer vision, computer graphics, and build-in software design. Image rotation is one of the most important topics for devices such as scanners. In fact there are a number of patents such as [3] proposing a method for rotating images so that the number of disc accesses is minimized and [4] using JPEG compression. Unfortunately, as far as the authors know, there are no theoretical results on this topics.

This paper is organized as follows. In Section 2 we give a mathematical description of our problem after preparing necessary notations and definitions. Then, in Section 3 we present a condition to determine whether interpolation at a given pixel is reliable or not only using local geometric information. Using the condition, we give an in-place algorithm for correcting a rotated subimage without using any extra working storage. We conclude the paper together with some open problems.

2 Problem Definition

In this section we formulate a problem mathematically. An input is an image which contains a subimage rotated by some angle θ. We assume that the rotation angle is a part of our input. Furthermore, for simplicity of argument we assume that the document is rotated in a counter-clockwise way. Rotation in the opposite direction can be dealt with in a symmetric manner.

Refer to Figure 1. The leftmost one is an image taken by a scanner. A document part in the figure is rotated. Given such a rotated image, we want to correct the rotation. The right figures ullustrate our strategy in this paper. We first execute interpolation at each pixel in the rotated subimage and store those

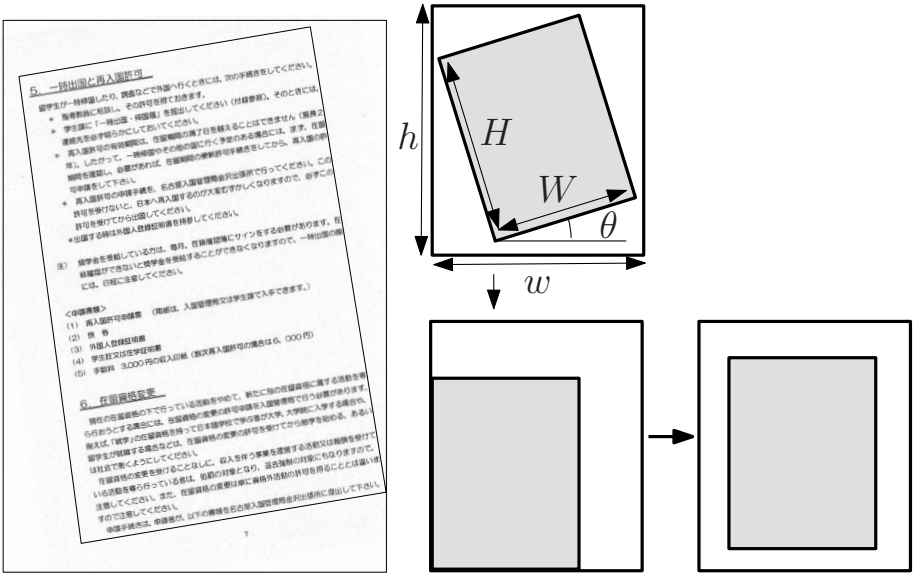


Fig. 1. An image containing a rotated subimage. A schematic illustration for correcting rotation is given in the right figures.

interpolated values over the input image. Finally, we shift the subimage to the center position.

2.1 Input Image and Rotated Subimages: G_{wh} and R_{WH}

Input image G consists of $h \times w$ pixels. Each pixel (x, y) is associated with an intensity level. The set of all those pixels (or lattice points in the xy-coordinate system) is denoted by $G^{\#}_{wh}$ and its bounding rectangle by G_{wh}.

Rotated subimage R consists of $h \times W$ pixels, which form a set $R^{\#}_{WH}$ of pixels (or lattice points in the XY-coordinate system). Intensity levels at each pixel (X, Y) is calculated by interpolation using intensity levels in the neighborhood.

2.2 Output Image and Location Function

An interpolation value calculated at a pixel $(X, Y) \in R^{\#}_{WH}$ in the rotated subimage is stored (or overwritten) at some pixel $s(X, Y) \in G^{\#}_{wh}$ in the original input image. The function $s(\ )$ determining the location is referred to as a location function. A simple function is $s(X, Y) = (X, Y)$ which maps a pixel (X, Y) in $R^{\#}_{WH}$ to a pixel (X, Y) in $G^{\#}_{wh}$. We could use different location functions, but this simple function seems best for row-major and column-major raster scans. So, we implicitly fix the function.

Then, an output image after correcting rotation is a range of the function. It is rather easy to move the output image to the center position of the original rectangle G_{wh} in an in-place manner.

2.3 Correspondence Between Two Coordinate Systems

Let (x_0, y_0) be the xy-coordinates of the lower left corner of the rotated document (more exactly, the lower left corner of the bounding box of the rotated subimage). Now, a pixel (X, Y) in $R^{\#}_{WH}$ is a point (x, y) in the rectangle G_{wh} with

$$x = x_0 + X\cos\theta - Y\sin\theta,$$
$$y = y_0 + X\sin\theta + Y\cos\theta.$$

The corresponding point (x, y) defined above is denoted by $p(X, Y)$.

2.4 Scan Order $\sigma(X, Y)$

Let σ be a scanning order over the pixels in $R^{\#}_{WH}$. It is a mapping from $R^{\#}_{WH}$ to a set of integers $\{0, 1, \ldots, WH-1\}$, that is, $\sigma(X, Y) = i$ means that the pixel (X, Y) is scanned in the i-th order. If σ is a row-major raster scan, $\sigma(X, Y) = X + Y \times W$ where $X = 0, \ldots, W-1$ and $Y = 0, \ldots, H-1$. A column-major raster order is symmetrically characterized by $\sigma(X, Y) = Y + X \times H$.

2.5 Window $N_d(x, y)$ for Interpolation

Following the scan order σ, we take pixels in the rotated image and for each pixel (X, Y) we compute an intensity value at (X, Y) by interpolation using intensity values of pixels in the neighborhood of the corresponding point $(x, y) = p(X, Y)$. There are a number of algorithms for interpolation. The simplest one called the nearest neighbor algorithm copies an intensity level from the nearest pixel. Linear interpolation performs interpolation by linear combination of intensity values at the four immediate neighbors. An algorithm using cubic polynomials for interpolation is called a cubic interpolation. Window used for the interpolation is denoted by $N_d(x, y)$, where d is a parameter to determine the size of the window. The value of d is 1 for linear interpolation and 2 for cubic interpolation. The window size of the nearest neighbor algorithm is at most 1, but only one point is used for interpolation. The window $N_d(x, y)$ is defined by

$$N_d(x, y) = \{(x', y') \in G^{\#}_{wh} | x' = \lfloor x \rfloor - d + 1, \ldots, \lfloor x \rfloor + d, y' = \lfloor y \rfloor - d + 1, \ldots, \lfloor y \rfloor + d\}.$$

The set $N_d(x, y)$ consists of at most $4d^2$ elements. We do not describe how linear or cubic interpolation is calculated.

2.6 Basic Interpolation Algorithm and Its Problem

The following is a basic algorithm for interpolation with a scan order σ and location function $s(\ )$.

Basic interpolation algorithm
(1) Scanning
for each $(X, Y) \in R^{\#}_{WH}$ in the scan order σ do

· Calculate the location $p(X, Y) = (x, y)$ in the xy-coordinate system.
· Execute interpolation at (x, y) using intensity levelsin the window $N_d(x, y)$.
· Store the interpolation value at a pixel $s(X, Y) \in G^{\#}_{wh}$ specified by the location function.

(2) Clear the margin
for each $(x, y) \in G^{\#}_{wh}$ do
if no interpolation value is stored at (x, y)
then the intensity level at (x, y) is set to *white*.

The basic algoritm above is simple and efficient. Unfortunately, it may lead to incorrect interpolations since to calculate an interpolation value at some pixel it may reuse intensity levels resulting from past interpolations. More precise description follows:

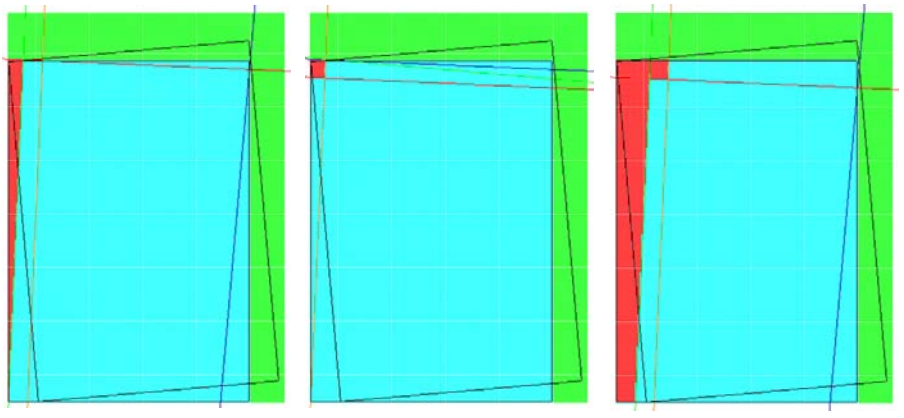
We say interpolation at $(X, Y) \in R^{\#}_{WH}$ is *reliable* if and only if none of the pixels in the window $N_d(x, y)$ keeps interpolation value. Otherwise, the interpolation is *unreliable*. "Unreliable" does not mean that the interpolation value at the point is incorrect. Consider an image of the same intensity level. Then, interpolation does not cause any change in the intensity value anywhere. Otherwise, if we use interpolated value for interpolation, the resulting value is different from the true interpolation value. We use the terminology "unreliable" in this sense. A pixel (X, Y) is called *reliable* if interpolation at (X, Y) is reliable and *unreliable* otherwise.

Figure 2 shows how frequently and where unreliable interpolations occur. In these figures those internal images are rotated counterclockwisely by degrees 5 without any left or bottom margins. When we scan the images by a usual row-major raster order with $d = 1$ (window size), those unreliable interpolations occur consecutively near the left boundary. If we use a column-major raster order instead, then we have mush less unreliable pixels as shown in (b) in the figure. If we make the window size d larger then the more unreliable pixels we have. The figure in (c) shows the case for $d = 2$.

3 Lazy Interpolation and Local Reliability Test

An idea to avoid such incorrect interpolation is to find all unreliable pixels and keep their interpolation values somewhere in a region which is not used for output image. In the following algorithm we use a queue to keep such interpolation values.

[**Lazy Interpolation**]
Q: a queue to keep interpolation values at unreliable pixels.
for each pixel $(X, Y) \in R^{\#}_{WH}$ in the order σ do
if (X, Y) is unreliable
then push the interpolation value at (X, Y) into the queue Q.
for each pixel $(X, Y) \in R^{\#}_{WH}$ in the order σ do
Calculate the interpolation value at (X, Y) and store the value at the pixel



(a) (b) (c)

Fig. 2. Distribution of unreliable pixels colored red (dark, if no color is available): image size = 234×170, rotation angle = 5 degrees counterclockwisely. (a) Row-major raster with $d = 1$, (b) column-major raster with $d = 1$, and (c) row-major raster with $d = 2$.

$s(X, Y) \in G^{\#}_{wh}$.
for each pixel $(X, Y) \in R^{\#}_{WH}$ in the order σ do
  if (X, Y) is unreliable then
    pop a value up from the queue Q and store the value at the pixel $s(X, Y)$.

Here are two problems. One is how to implement the queue. The other is how to check unreliability of a pixel. It should be remarked that both of them must be done without using any extra working storage.

Suppose we scan pixels in a rotated subimage $R^{\#}_{WH}$ according to a scan order σ and interpolation using a window of size d around each point (X, Y) is calculated and stored at an array element $s(X, Y)$ specified by the location function. Now we can define another sequence τ to determine an order of all pixels in $G^{\#}_{wh}$ to receive interpolated values. That is, the function τ is defined so that

$$\tau(s(X, Y)) = \sigma(X, Y)$$

holds for any $(X, Y) \in R^{\#}_{WH}$. Since rotated subimage is smaller than the original image, some pixels in the original image are not used for output image. That is, there are pixels (x, y) in $G^{\#}_{wh}$ such that there is no (X, Y) in $R^{\#}_{WH}$ with $(x, y) = s(X, Y)$. For such pixels (x, y) we define $\tau(x, y) = WH$. More precisely, τ is a mapping from $G^{\#}_{wh}$ to $\{0, 1, \ldots, WH\}$ such that

$\tau(x, y) = i < WH$ if i-th computed interpolation value is stored at (x, y) in $G^{\#}_{wh}$,
$\tau(x, y) = WH$ if no interpolation value is stored at (x, y).

Then, interpolation at (X, Y) is reliable in the sense defined in the previous section if none of the pixels in the window does not keep interpolated value, that is,

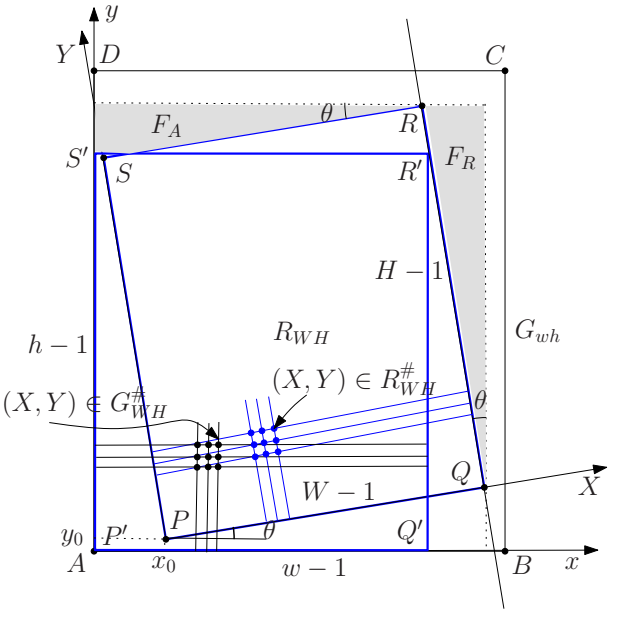


Fig. 3. Two rectangles G_{wh} and R_{WH}

$$\tau(x,y) \geq \sigma(X,Y) \text{ for each } (x,y) \in N_d(p(X,Y)).$$

This condition referred to as the reliability condition.

Figure 3 illustrates two rectangles, $ABCD$ for G_{wh} and $PQRS$ for R_{WH}.

3.1 Row-Major Raster Scan for Counterclockwise Rotation

Consider a simple case where σ is a row-major raster scan. Let $(x,y) = p(X,Y)$, that is,

$$x = x_0 + X\cos\theta - Y\sin\theta,\ \ y = y_0 + X\sin\theta + Y\cos\theta.$$

If we order those pixels in the interpolation window of size d around (x,y) in the order of receiving interpolation values, then the first point is $(\lfloor x\rfloor - d + 1, \lfloor y\rfloor - d - 1)$ because interpolation values are also filled in $G^{\#}_{wh}$ in the same row-major raster order (restricted to the part $0 \leq x < W$ and $0 \leq y < H$). If the first part has not received any interpolation value, that is, if $\tau(\lfloor x\rfloor - d + 1, \lfloor y\rfloor - d - 1) \geq \sigma(X,Y)$, then the pixel (X,Y) is reliable. Otherwise, it is unreliable. By the definition of σ and τ, we have a simpler expression of the condition.

Lemma 1. [Local Reliability Condition] *Assuming a row-major raster order for σ and τ, pixel $(X,Y) \in R^{\#}_{WH}$ is unreliable if and only if*
(1) $x_0 + X\cos\theta - Y\sin\theta - d + 1 < X$ *and* $y_0 + X\sin\theta + Y\cos\theta - d < Y$, *or*
(2) $x_0 + X\cos\theta - Y\sin\theta - d + 1 < W$ *and* $y_0 + X\sin\theta + Y\cos\theta - d + 1 < Y$.

Proof. By the condition stated above, a pixel (X, Y) is unreliable if and only if
(1) $\lfloor x_0 + X\cos\theta - Y\sin\theta \rfloor - d + 1 \le X - 1$ and $\lfloor y_0 + X\sin\theta + Y\cos\theta \rfloor - d + 1 \le Y$, or
(2) $\lfloor x_0 + X\cos\theta - Y\sin\theta \rfloor - d + 1 \le W - 1$ and $\lfloor y_0 + X\sin\theta + Y\cos\theta \rfloor - d + 1 \le Y - 1$.

Let a and b be two arbitrary positive real numbers. Then, $\lfloor a \rfloor \ge \lfloor b \rfloor$ holds if and only if $a \ge \lfloor b \rfloor$. Also, $\lfloor a \rfloor \le \lfloor b \rfloor$ holds if and only if $a < \lfloor b \rfloor + 1$. Using these inequalities, the above condition can be restated as in the lemma. □

An importance of Lemma 1 is that it suggests a way of testing reliability of interpolation at each pixel without using any working array. That is, it suffices to check the two conditions in the lemma.

By Lemma 1, a pixel (X, Y) is unreliable if and only if

$$(1)\ Y > -\tfrac{1-\cos\theta}{\sin\theta}X + \tfrac{x_0-d+1}{\sin\theta} \text{ and } Y > \tfrac{\sin\theta}{1-\cos\theta}X + \tfrac{y_0-d}{1-\cos\theta} \text{ or}$$
$$(2)\quad Y > \tfrac{\cos\theta}{\sin\theta}X - \tfrac{W-x_0+d-1}{\sin\theta} \text{ and } Y > \tfrac{\sin\theta}{1-\cos\theta}X + \tfrac{y_0-d+1}{1-\cos\theta}.$$

By L_1, L_2, L_3 and L_4 we denote the four lines above:

$$L_1:\ Y = -\tfrac{1-\cos\theta}{\sin\theta}X + \tfrac{x_0-d+1}{\sin\theta},\quad L_2:\ Y = \tfrac{\sin\theta}{1-\cos\theta}X + \tfrac{y_0-d}{1-\cos\theta},$$
$$L_3:\ Y = \tfrac{\cos\theta}{\sin\theta}X - \tfrac{W-x_0+d-1}{\sin\theta},\quad L_4:\ Y = \tfrac{\sin\theta}{1-\cos\theta}X + \tfrac{y_0-d+1}{1-\cos\theta}.$$

Then, a pixel (X, Y) is unreliable if and only if the point (X, Y) is above the two lines L_1 and L_2 or above the two lines L_3 and L_4.

3.2 Column-Major Raster Scan for Counterclockwise Rotation

How about a column-major raster order instead of row-major order? By similar arguments we have a similar observation.

Lemma 2. *Assuming a column-major raster order for σ and τ, a pixel $(X, Y) \in R^{\#}_{WH}$ is unreliable if and only if*
(1') $x_0 + X\cos\theta - Y\sin\theta - d < X$ and $y_0 + X\sin\theta + Y\cos\theta - d + 1 < Y$, or
(2') $x_0 + X\cos\theta - Y\sin\theta - d + 1 < X$ and $H > y_0 + X\sin\theta + Y\cos\theta - d + 1 \ge Y$.

By Lemma 2, a pixel (X, Y) is unreliable if and only if

$$(1')\ Y > -\tfrac{1-\cos\theta}{\sin\theta}X + \tfrac{x_0-d}{\sin\theta} \text{ and } Y > \tfrac{\sin\theta}{1-\cos\theta}X + \tfrac{y_0-d+1}{1-\cos\theta} \text{ or}$$
$$(2')\ Y > -\tfrac{1-\cos\theta}{\sin\theta}X + \tfrac{x_0-d+1}{\sin\theta} \text{ and } Y < -\tfrac{\sin\theta}{\cos\theta}X + \tfrac{H-y_0+d-1}{\cos\theta}.$$

By L'_1, L'_2, L'_3 and L'_4 we denote the four lines above:

$$L'_1:\quad Y = -\tfrac{1-\cos\theta}{\sin\theta}X + \tfrac{x_0-d}{\sin\theta},\quad L'_2:\ Y = \tfrac{\sin\theta}{1-\cos\theta}X + \tfrac{y_0-d+1}{1-\cos\theta},$$
$$L'_3:\ Y = -\tfrac{1-\cos\theta}{\sin\theta}X + \tfrac{x_0-d+1}{\sin\theta},\quad L'_4:\ Y = -\tfrac{\sin\theta}{\cos\theta}X + \tfrac{H-y_0+d-1}{\cos\theta}.$$

Figures 4 (a) and (b) depict the four lines and the region of unreliable pixels bounded by them for each of row-major and column-major raster orders.

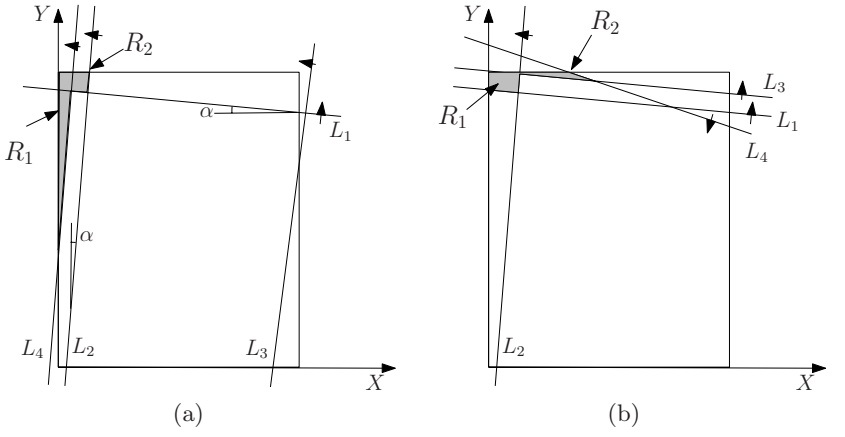


Fig. 4. Regions of unreliable pixels, (a) for row-major raster order, and (b) for column-major raster order

3.3 Lazy Interpolation for $d = 1$

Now we know how to detect possibility of unreliable pixel each in constant time. If each pixel is reliable, we just perform interpolation. Actually, if the bottom margin y_0 is large enough, then the location $s(X, Y)$ to keep interpolation value is far from a point (X, Y) and thus it does not affect interpolation around the point. Of course, if the window size d is large, then interpolations become more frequently unreliable.

Here we present an in-place algorithm for correcting rotation. For the time being we shall concentrate ourselves in the simpler case $d = 1$. A key to our algorithm is the local test on reliability. In our algorithm we scan $R^{\#}_{WH}$ three times. In the first scan, we check whether (X, Y) is a reliable pixel or not each in constant time. If it is not reliable, we calculate interpolation value and store it somewhere in $G^{\#}_{wh}$ using pixels outside the rectangle determining the output image. Such a region is called a *refuge*.

In-place algorithm for correcting rotation

Phase 1: For each $(X, Y) \in R^{\#}_{WH}$ check whether a pixel (X, Y) is reliable or not. If it is not, then calculate interpolation there and store the value in the refuge F.

Phase 2: For each $(X, Y) \in R^{\#}_{WH}$ calculate interpolation there and store the value at $(X, Y) \in G^{\#}_{wh}$.

Phase 3: For each $(X, Y) \in R^{\#}_{WH}$ check whether interpolation at (X, Y) is reliable or not. If it is not, then update the value at $(X, Y) \in G^{\#}_{wh}$ by the interpolation value stored in the refuge F.

The algorithm above works correctly when $d = 1$. The most important is that the total area of refuge available is always greater than the total number of unreliable pixels.

Theorem 1. *The algorithm above correctly computes interpolations for row-major and column-major raster scans with the location function* $s(X, Y) = (X, Y)$.

Proof. We do not prove correctness of the algorithm due to space limit. We only prove that we can always find a refuge F sufficiently large. Because of similarity we only prove the theorem for the row-major raster scan.

As described earlier, the region of unreliable pixels is divided into two regions, one bounded by the two lines L_1 and L_2, and the other by L_4 and the left boundary of R_{WH}. The two regions are denoted by R_1 and R_2 in this order, as shown in Figure 4.

We have two rectangles G_{wh} corresponding to an input image and R_{WH} to a rotated subimage. With the location function $s(X, Y) = (X, Y)$, the output image is determined by rotating R_{WH} clockwisely by the angle θ and translating it so that the lower left corner coincides with the lower left corner of G_{wh}. Drawing the horizontal line through the upper right corner and vertical line through the lower right corner of R_{WH}, we have two regions F_L and F_A, as shown in Figure 3, which can be used as refuge. In other words, we can store any values there without affecting correct interpolations to be output.

To ease the proof we assume that there is no margin between the two rectangles G_{wh} and R_{WH}, that is, the four corners of R_{WH} all lie on the boundary of G_{wh}. In this case we have $x_0 = (H-1)\sin\theta$ and $y_0 = 0$. Since $d = 1$, the line L_1 passes through $(0, H-1)$ and L_4 does $(0, 0)$. The angle α between the line L_4 and the vertical line is smaller than θ because

$$\tan(\alpha) = \frac{1-\cos\theta}{\sin\theta} < \tan\theta.$$

Thus, the area of the region (R_1 in Figure 4 (a)) bounded by L_4 and the left boundary is smaller than the refuge F_R bounded by the line RQ and the right boundary of G_{wh} (see Figure 3).

By the same reason we can also prove that the area of the region R_2 bounded by L_1 and L_2 is smaller than that of the region F_A above the line SR in Figure 3. This completes the proof. □

3.4 Lazy Interpolation for $d = 2$

With a larger window of size $d \geq 2$ the algorithm above does not work due to insufficient area of the refuge. Fortunately, if the lower margin, y_0, is at least $d-1$, then the lazy interpolation for the column-major raster works correctly. When $y_0 = d-1$ and $d \geq 2$, the unreliable region is the union of the two regions R_1 above L'_1 and L'_2 and R_2 above L'_3 and below L'_4. The line L'_2 passes through the origin, we can use the right refuge F_R as before for R_1.

What about the region R_2 bounded by L'_3 and L'_4? The line L'_4 is parallel to the horizontal side of the rectangle G_{wh} and the line L'_3 has smaller slope than the upper side of the rotated rectangle. Hence, the angle between L'_3 and L'_4 is smaller than θ. This implies that the region R_2 bounded by L'_3 and L'_4 has smaller area than the upper refuge F_A. See Figure 5 for illustration.

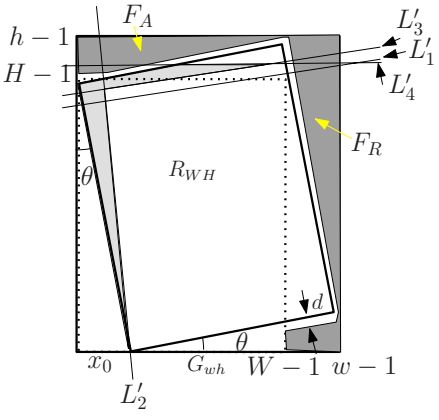


Fig. 5. The region of unreliable pixels and right and top refuges F_R and F_A

Unfortunately we cannot use the algorithm above for a larger window, $d = 2$ since we have so many unreliable pixels even in the case. The idea here is to use a queue to store interpolation values at unreliable pixels and pop them up whenever storing them does not cause any harm for interpolations. The region outside the rotated image and the output image, shown in Figure 5, can be used for the purpose.

Assume a row-major raster order. Suppose we are going to calculate interpolation at pixels in a row Y. Then, the pixel values below the row $\lfloor y_0 + Y \cos\theta \rfloor - d$ (including the row) are never used for interpolations. Let us call the row the high limit for Y. If it is greater than the previous high limit, i.e., $\lfloor y_0 + (Y-1)\cos\theta \rfloor - d$, then we can safely store interpolation values at the row. This observation leads to the following algorithm.

In-place algorithm 2 for correcting rotation

Q = a queue containing interpolated values, using the region in the refuge.
for each row $Y = 0$ to $H - 1$ do
  for each $X = 0$ to $W - 1$
    if (X, Y) is unreliable
      then push the interpolation value at (X, Y) into the queue Q.
  if $\lfloor y_0 + Y\cos\theta \rfloor - 2 > \lfloor y_0 + (Y-1)\cos\theta \rfloor - 2$
    then $Y' = \lfloor y_0 + Y\cos\theta \rfloor - 2$.
    for each $X = 0$ to $W - 1$
      if (X, Y) is unreliable
        then store the value popped from Q at $s(X, Y)$.
        else calculate interpolation value at (X, Y) and store it at $s(X, Y)$.

Unfortunately, no formal proof has not been obtained for correctness of the algorithm above. However, it has caused no problem for practical applications.

4 Concluding Remarks and Future Works

In this paper we have presented in-place algorithms for correcting rotation of a subimage contained in an image using interpolation. We have shown that as long as interpolation is implemented by linear interpolation algorithm we can always correct any rotation without using any extra working array. Correctness proof for a larger window used for cubic interpolation has been left as an open problem.

In this paper we considered two scan orders, row-major and column-major raster orders. Many other scan orders are possible. In addition to row- and column major raster scans we could scan an image at any angle. One of promising scans is the following: First, find a rotation angle θ. Then, round it to an anlge θ' defined by two pixels in a rotated subimage. Using this approximate angle, we can scan all of pixels in the rotated subimage without any extra working storage.

It is interesting to evaluate and compare those scan orders by the number of unreliable pixels. The best scan order may depend on margins. In our experience, if the bottom margin is greater than the left margin then the row-major raster is better than the column-major one. If the left margin is larger than the bottom margin, the column-major raster outperforms row-major raster. But there is no formal proof.

References

1. Asano, T., Katoh, N.: Variants for Hough Transform for Line Detection. Computational Geometry: Theory and Applications 6, 231–252 (1996)
2. Duda, R.O., Hart, P.E.: Use of the Hough Transformation to Detect Lines and Curves in Pictures. Comm. ACM 15, 11–15 (1972)
3. Kermisch, D.: Rotation of digital images, United States Patent, 4545069 (1985)
4. Micco, F.A., Banton, M.E.: Method and apparatus for image rotation with reduced memory using JPEG compression, United States Patent, 5751865 (1998)

Higher Order Voronoi Diagrams of Segments for VLSI Critical Area Extraction

Evanthia Papadopoulou

IBM T.J. Watson Research Center, Yorktown Heights, NY 10598, USA
Athens University of Economics and Business, Athens 10434, Greece
evanthia@acm.org

Abstract. We address the problem of computing critical area for opens in a circuit layout in the presence of multilayer loops and redundant interconnects. The extraction of critical area is a fundamental problem in VLSI yield prediction. We first model the problem as a graph problem and we solve it efficiently by exploiting its geometric nature. We introduce the *opens Voronoi diagram of polygonal objects*, a generalization of Voronoi diagrams based on concepts of higher order Voronoi diagrams of segments. Higher order Voronoi diagrams of segments had not received much attention in the literature. The approach expands the Voronoi critical area computation paradigm [1,2,3] with the ability to accurately compute critical area for missing material defects even in the presence of loops and redundant interconnects spanning over multiple layers.

1 Introduction

Catastrophic yield loss of integrated circuits is caused to a large extent by random particle defects interfering with the manufacturing process resulting in functional failures such as open or short circuits. All yield models for random manufacturing defects focus on critical area, a measure reflecting the sensitivity of the design to random defects during manufacturing (e.g. [4,5,6]). Reliable critical area extraction is essential for today's IC manufacturing especially when DFM, i.e., design for manufacturability, initiatives are under consideration.

The critical area of a circuit layout on a layer A is defined as

$$A_c = \int_0^\infty A(r)D(r)dr$$

where $A(r)$ denotes the area in which the center of a defect of radius r must fall in order to cause a circuit failure and $D(r)$ is the density function of the defect size. $D(r)$ has been estimated as $D(r) = r_0^2/r^3$, where r_0 is some minimum optically resolvable size. Critical area analysis is typically performed on a per layer basis and results are combined to estimate total yield.

In this paper we focus on critical area extraction for *open faults* resulting from broken interconnects generalizing upon the results of [2]. Opens are net-aware, that is, a defect is considered a fault only if it actually *breaks* a net. A net is

T. Tokuyama (Ed.): ISAAC 2007, LNCS 4835, pp. 716–727, 2007.

said to be *broken* if there exist terminal points that get disconnected. In order to increase design reliability and reduce the potential for open circuits designers are increasingly introducing redundant interconnects creating interconnect loops that may span over a number of layers (see e.g. [7]). Redundant interconnects reduce the potential for open faults at the expense of increasing the potential for shorts. The ability to perform trade-offs is important requiring accurate critical area computation for both.

In this paper we first model the problem as a graph problem and we solve it efficiently by exploiting the geometric nature of it. We formulate the *opens Voronoi diagram*, a generalization of Voronoi diagrams based on concepts of higher order Voronoi diagrams of segments. To the best of our knowledge higher order Voronoi diagrams of segments have not received much attention in the literature with the recent exception of [8] for the farthest segment Voronoi diagram. Once the opens Voronoi diagram is available the entire critical area integral for opens can be computed analytically in linear time similarly to [1,9,2].

For computational simplicity we have adapted the L_∞ metric throughout this paper [9]. This is consistent with the common practice of modeling defects as squares to facilitate critical area computation. For simplicity figures are depicted in Manhattan geometry, however, the method is general and it is applicable to layouts of arbitrary geometry as well as other metrics of potential interest such as the Euclidean or the k-gon metric. The algorithms presented in this paper have been integrated into the IBM Voronoi Critical Area Analysis tool (*Voronoi CAA*) which is used extensively by IBM manufacturing for the prediction of yield. For results on the industrial use of our tool and comparisons with previously available tools see [10].

2 Review of L_∞ Voronoi Diagrams for Opens

The Voronoi diagram of a set of polygonal sites in the plane is a partitioning of the plane into regions, one for each site, called *Voronoi regions*, such that the Voronoi region of a site s is the locus of points closer to s than to any other site. The Voronoi region of s is denoted as $reg(s)$ and s is called the *owner* of $reg(s)$. The boundary that borders two Voronoi regions is called a *Voronoi edge*, and consists of portions of *bisectors* between the owners of the neighboring cells. The bisector of two polygonal objects (such as points, segments, polygons) is the locus of points equidistant from the two objects. The point where three or more Voronoi edges meet is called a *Voronoi vertex*. The combinatorial complexity of the Voronoi diagram is linear in the number and complexity of the sites. In the interior of a simple polygon the Voronoi diagram is also called *medial axis*[1]. Any point p on the boundary of $reg(s)$ is weighted by $w(p) = d(p, s)$. The disk D centered at p of radius $w(p)$ is *empty*, that is, no site intersects the interior of D.

The L_∞ distance is used throughout this paper. The L_∞ distance between two points $p = (x_p, y_p)$ and $q = (x_q, y_q)$ is $d(p, q) = \max\{|x_p - x_q|, |y_p - y_q|\}$. In

[1] There is a minor difference in the definition which we ignore in this paper (see [11]).

the presence of additive weights, the (weighted) distance is $d_w(p,q) = d(p,q) + w(p) + w(q)$, where $w(p)$ and $w(q)$ denote the weights of points p, q respectively. In case of a weighted line l, the (weighted) distance between a point t and l is $d_w(t,l) = \min\{d(t,q)+w(q), \forall q \in l\}$. The (weighted) bisector between polygonal elements s_i, s_j is $b(s_i,s_j) = \{y \mid d_w(s_i,y) = d_w(s_j,y)\}$. In this paper we use the term *core element*, i.e., *core segment* and *core point*, to denote a portion of interest along a bisector (such as a medial axis segment, a Voronoi edge or a Voronoi vertex). Fig. 1 illustrates examples of core segments. The endpoints and the open line segment portion of a core segment are always differentiated and they are treated as distinct entities.

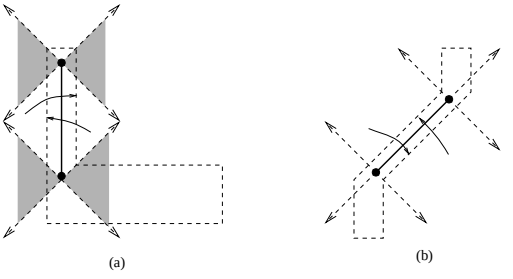


Fig. 1. The regions of influence of the core elements of a core segment

In L_∞ core segments can be treated as additively weighted ordinary segments. Let s be a core segment induced by the polygonal elements e_l, e_r, that is, s is portion of bisector $b(e_l, e_r)$. Every point p along s is weighted with $w(p) = d(p,e_l) = d(p,e_r)$. The 45° rays[2] emanating from the endpoints of s partition the plane into the regions of influence of either the open core segment portion or the core endpoints. (In the Euclidean metric the corresponding rays would be rays perpendicular to e_l and e_r). Fig.1 illustrates the partitioning of space induced by a core segment in the L_∞ metric. Shaded regions are equidistant from both the core endpoint and the open core segment. In this paper equidistant regions are always assigned to the core endpoint. In the region of influence of a core point p distance is measured in the ordinary weighted sense, that is, for any point t, $d_w(t,p) = d(t,p) + w(p)$. In the region of influence of an open core segment s distance in essence is measured according to the farthest polygonal element defining s, that is, $d_w(t,s) = d(t,e_l)$ where e_l is the polygonal element at the opposite side of the line through s than t (indicated by arrows in Fig.1). In L_∞ this is equivalent to the ordinary weighted distance from s. The bisector between two core elements, and therefore the Voronoi diagram of a set of core elements, can now be defined as usual. The (weighted) Voronoi diagram of *core* medial axis segments was introduced in [2] as a solution to the critical area computation problem for a simpler notion of an open based solely on geometric information called *break*.

[2] A 45° ray is a ray of slope ±1.

3 A Graph Representation for Nets

From a layout perspective a net N is a collection of interconnected shapes spanning over a number of layers. The portion of N on a given layer X is denoted as $N_X = N \cap X$ and consists of a number of connected components interconnected through different layers. Every connected component is a collection of overlapping polygons that can be unioned into a single shape (a simple one or one with holes). Some of the shapes constituting net N are designated as *terminal shapes* representing the entities that the net must interconnect. A net remains *functional* as long as all terminal shapes comprising the net remain interconnected. Otherwise the net is said to be *broken*. Fig. 2(a) illustrates a simple net N spanning over two metal layers, say M1 and M2, where M2 is illustrated shaded. The two contacts illustrated as black squares have been designated as terminal shapes. In Fig. 2(b) defects that create opens are illustrated as dark squares and defects that cause no fault are illustrated hollow in dashed lines. Note that hollow defects do break wires of layer M1 but they do not create an open as they do not break net N.

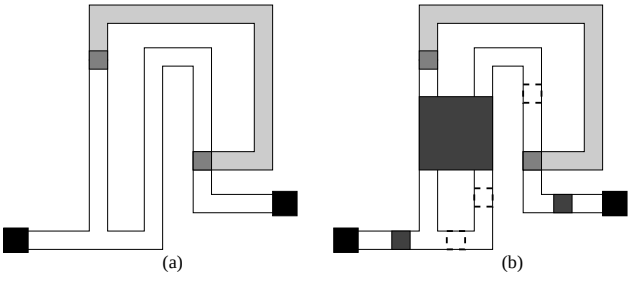


Fig. 2. (a) A net N spanning over two layers. (b) Dark defects create opens while transparent defects cause no faults.

We define a compact graph representation for N, denoted $G(N)$, as follows. There is a graph node for every connected component of N on a conducting layer. A node containing terminal shapes is designated as a terminal node. Two graph nodes are connected by an edge iff there exists at least one contact or via connecting the respective components of N. To build $G(N)$ some net extraction capability needs to be available. Net extraction is a well studied topic beyond the scope of this paper. For the purposes of this paper we assume that $G(N)$ can be available for any net.

To perform critical area computation on a layer A we derive the *extended graph* of N on layer A, denoted as $G(N, A)$, that can be obtained from $G(N)$ by expanding all components of N_A by their medial axis. For every via or contact introduce an approximate point along the medial axis representing that via or contact, referred to as a *via-point*, and a graph edge connecting the via-point with the node of the connecting component of N. If a contact or via has been

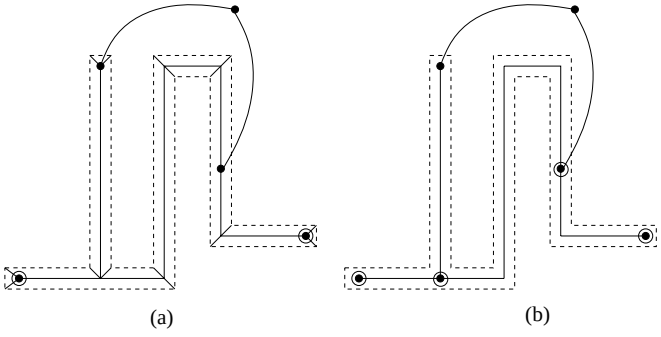


Fig. 3. The net graph of Fig. 2 before (a) and after (b) cleanup of trivial parts

designated as terminal shape, designate also the corresponding via point as terminal. In the presence of via clusters we can keep only one via point representing the entire cluster. Any portion of the medial axis induced by edges of terminal shapes is also identified as terminal. Fig. 3a illustrates $G(N, A)$, where $A = M1$, for the net of Fig. 2. Terminal points are indicated by hollow circles. Dashed lines represent the original M1 polygon and are not part of $G(N, M1)$.

Given $G(N, A)$ we can detect *biconnected components, bridges* and *articulation points* using *depth-first search* (DFS) as described in [12,13]. For our problem we only maintain some additional terminal information to determine whether the removal of a vertex or edge actually *breaks* $G(N, A)$, i.e., whether it disconnects $G(N, A)$ leaving terminals in both sides. For this purpose we chose the root of the DFS tree to be a terminal node or terminal point and at every node i of the DFS tree we keep a flag indicating whether the subtree rooted at i contains a terminal point. Any bridges or any articulation points whose removal does not disconnect terminals of $G(N, A)$ are called *trivial*. Similarly any biconnected component incident to only trivial articulation points that contains no terminal points is called trivial. Trivial bridges, trivial articulation points and trivial biconnected components can be easily determined and removed from the graph with no effect on the net connectivity regarding opens. In the following we assume that $G(N, A)$ has been cleaned up from all trivial parts, and thus, the removal of any bridge or any articulation point results in a fault. Fig. 3(b) illustrates the net graph after the cleaning of all trivial parts. Hollow circles indicate terminal and articulation points. The graph has exactly one bi-connected component.

4 Modeling Net-Aware Opens

In this section we formalize the definition of a net-aware open.

Definition 1. *A* minimal open *is a defect D that breaks a net N and D has minimal size, that is, if D is shrunk by $\epsilon > 0$ then D no longer breaks N. D breaks N if any two terminal shapes of N get disconnected or if a terminal shape*

itself gets destroyed. A minimal open is called strictly minimal *if it contains no other minimal open in its interior. An* open *is any defect entirely covering a minimal open. The size of a minimal open centered at a point t is called the* critical radius *of t.*

Definition 2. *The center point of an open D is called a* generator point *for D and it is weighted with the size (radius) of D. A segment formed as a union of generator points is called a* generator segment *or simply a generator.*

Definition 3. *The* core *of the extended net graph $G(N, A)$ on layer A, denoted $core(N, A)$, is the set of all medial axis vertices, including articulation, via, and terminal points, and all medial axis edges, except the standard-45° edges*[3]*. $G(N, A)$ is assumed to have been cleaned up from any trivial components, trivial bridges, or trivial articulation points. The union of $core(N, A)$ for all nets N is denoted as $core(A)$.*

A core segment is assumed to consist of three distinct core elements: two endpoints and an open line segment. Given a net N, $core(N, A)$ induces a unique decomposition of the portion of N on layer A into well defined wire segments. In particular, any core element s induces a wire segment $R(s) = \cup_{p \in s} R(p)$, where $R(p)$ denotes the disk (the square in L_∞) centered at core point p having radius $w(p)$. Those wire segments may overlap and their union reconstructs N_A (excluding the trivial portions of N_A). In Fig. 3b all depicted medial axis vertices and segments constitute $core(N, A)$. The dark shaded disks of Fig. 2 are strictly minimal opens.

Definition 4. *A defect D is classified as order $k, k \geq 1$, if D overlaps k wire segments as induced by k distinct core elements of $core(A)$. The center point of D is classified as a kth order generator, $k \geq 1$. In case of core elements equidistant (in weighted sense) from the center of D, a kth order defect is allowed to overlap more than k wire segments.*

Definition 5. *A* cut *is a collection of core elements $C \subset core(N, A)$, such that $G(N, A) - C$ is disconnected leaving non-trivial articulation or terminal points in at least two different sides. A cut C of k elements is called minimal if $C - \{c\}$ is not a cut for any element $c \in C$*

Lemma 1. *The set of 1st order generators for strictly minimal opens on layer A consists exactly of the all the bridges, terminal edges, articulation points, and terminal points of $G(N, A) \cap core(N, A)$ for every net N.*

5 A Higher Order Voronoi Digram Modeling Opens

The Voronoi diagram for opens on layer A, referred to as the *opens Voronoi diagram*, is a subdivision of the layer into regions such that each region reveals

[3] The term standard-45° refers to portions of bisectors of axis parallel lines of slope ± 1.

the *critical radius* for opens for every point in that region. Recall that the critical radius of a point t, $r_c(t)$, is the size of the smallest defect centered at t causing a circuit failure. A circuit failure here corresponds to an open. For any point t in a region $reg(h)$ of the opens Voronoi diagram the critical radius of t should be derived as a distance function from the owner h, specifically $r_c(t) = d_w(t, h)$ for $h \in core(A)$. In the following we formulate the opens Voronoi diagram as a higher order Voronoi diagram of core segments.

Consider the (weighted) Voronoi diagram of all core elements on layer A, denoted as $V(A)$. If there are no loops associated with layer A then all elements of $core(A)$ must be 1st order generators and $V(A)$ must provide the opens Voronoi diagram on layer A (see also [2]). Fig. 4 illustrates $V(A)$ for the net graph of Fig. 2. To model opens appropriately we follow some additional conventions for $V(A)$ as follows: A region equidistant from a core segment and its endpoint is always assigned to the endpoint. All regions of 1st order generators are colored red. Coloring a region red indicates that the critical radius of every point in the region is determined by the owner of that region.

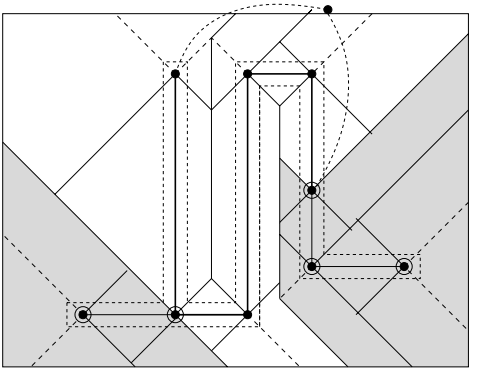


Fig. 4. The Voronoi diagram $V(A)$ for a net N on layer A

Let us now define the order-k Voronoi diagram of layer A, denoted as $V^k(A)$. For $k = 1$, $V^k(A) = V(A)$. A non-red region of $V^k(A)$ is a locus of points with the same k nearest neighbors (in a weighted sense) among the core elements in $core(A)$. A red region of $V^k(A)$ is a locus of points with the same r, $1 \leq r \leq k$, nearest neighbors among the core elements in $core(A)$, such that the set C of those r core elements constitutes a minimal cut for some net N. If $|C| > 1$ the red Voronoi region $reg(C)$ is further subdivided into finer subregions by the farthest Voronoi diagram of C, denoted $V_f(C)$. Fig. 5 illustrates $V^2(A)$ for the net of our example. Voronoi regions of $V^2(A)$ are illustrated in solid lines and red regions are illustrated shaded. The thick dashed lines indicate $V_f(C)$ for cuts $C, |C| = 2$.

There are two types of red regions in $V^k(A)$: those that are expanded red regions of $V^{k-1}(A)$, referred to as *old red regions*, and *new red regions* of cuts

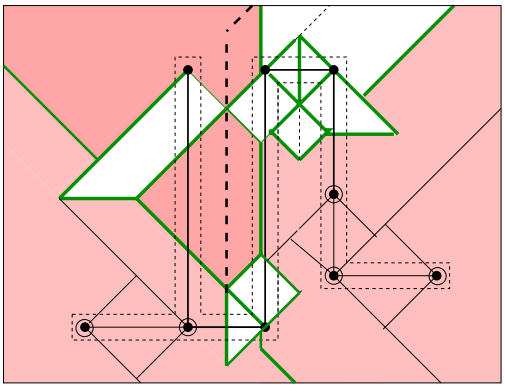


Fig. 5. The 2nd order Voronoi diagram $V^2(A)$

determined in $V^k(A)$. Clearly any red region of $V^k(A)$ remains red in $V^{k+1}(A)$. In Fig. 5 new red regions are illustrated darker. The thick dashed axis-parallel segments in the new red regions of Fig. 5 are the 2nd order generators for minimal opens.

Theorem 1. *The Voronoi diagram for opens on layer A is the minimum order-k Voronoi diagram of core(A), $V^k(A)$, such that all regions of $V^k(A)$ are colored red. Any region $reg(H)$ such that H consists of more than one core element is further subdivided into finer regions by $V_f(H)$, the farthest Voronoi diagram of H. The critical radius for any point t in a Voronoi region $reg(H)$ is $r_c(t) = \max\{d_w(t,h), h \in H\}$. In particular, if t is in the subregion $reg(h) \subset reg(H) \cap V_f(H)$ then $r_c(t) = d_w(t,h)$.*

Corollary 1. *The higher order generators for strictly minimal opens on layer A are exactly the farthest Voronoi edges and vertices (except the standard-45° Voronoi edges) of the opens Voronoi diagram on layer A constituting the farthest Voronoi subdivisions in the Voronoi region of any cut C of size $|C| > 1$.*

To the best of our knowledge higher order Voronoi diagrams of segments have not received much attention in the literature unlike higher order Voronoi diagrams of points, see e.g. [11,14,15]. The problem can have different flavors depending on whether segments are treated as closed entities or whether the open portions of segments are treated as distinct from their endpoints. In this paper we only deal with the latter interpretation as this is the one modeling our application.

5.1 Computing the Opens Voronoi Diagram

To obtain the Voronoi diagram for opens we can adapt the simple iterative process to obtain higher order Voronoi diagrams of points (see e.g. [11]) as we are primarily interested in small values of k. The main difference with the standard

case of points is that an open core segment s does not exist in the region of its endpoint p, that is, s can not be considered as a higher order nearest neighbor in $reg(p)$. Furthermore, in L_∞, there can be regions equidistant from more than one element. As a result, unlike the Euclidean metric, the k-tuples owning two neighboring Voronoi regions may differ in more than one element.

We first obtain $V(A)$ by plane sweep modifying the algorithm of [9] to accommodate weights and the convention of assigning priority to endpoints as opposed to the open portion of segments. Note that weights are special ensuring no disconnected Voronoi regions. The latter convention can be accommodated either by modifying the original algorithm to include the bisectors (45° rays in L_∞) between the open portion of core segments and their endpoints, or after the original Voronoi diagram is constructed by drawing the additional bisectors in linear time. Intersections among the additional bisectors can be resolved arbitrarily. All Voronoi subregions associated with the same core point p are unified into a single Voronoi region for p. The properties and the asymptotic combinatorial complexity of the Voronoi diagram remain the same.

Let's now assume that $V^k(A), k \geq 1$, is available. We show how to compute $V^{k+1}(A)$. Let $reg(H)$ be a non-red region of $V^k(A)$ and let $s(H)$ denote the superset of H defined as H union all open core segments incident to the core points in H. Let $N(H)$ denote the union of all core elements owning Voronoi regions neighboring the regions of elements in $s(H)$. Compute the (weighted) L_∞ Voronoi diagram of $N(H) - s(H)$ and truncate it within the interior of $reg(H)$; this gives the $(k+1)$-order subdivision within $reg(H)$. Each $(k+1)$-order subregion within $reg(H)$ is attributed to a $(k+1)$-tuple $H \cup \{c\}$, where $c \in N(H) - s(H)$. Once the $(k+1)$-order subdivision has been performed within the non-red regions of $V^k(A)$ we remove the edges and vertices of $V^k(A)$ that are not part of $V^{k+1}(A)$, merge the incident $(k+1)$-order subregions of $V^k(A)$ into the $(k+1)$-order Voronoi regions of $V^{k+1}(A)$, and determine the red regions of $V^{k+1}(A)$. Unlike the standard higher order Voronoi diagram case, not all Voronoi edges of $V^k(A)$ are necessarily deleted from $V^{k+1}(A)$. In the following we give the details of this process.

Let $reg(H)$ be a non-red region of $V^k(A)$ and let c, $c \notin H$, be a core element inducing a $(k+1)$-order subregion in $reg(H)$. Let $reg(H \cup \{c\})$ denote the union of all $(k+1)$-order subregions of $V^k(A)$ owned by $H \cup \{c\}$. Recall that no $(k+1)$-order subdivision is performed within red regions of $V^k(A)$, that is, no portion of $reg(H \cup \{c\})$ is red other than possibly some bounding Voronoi edges. Any Voronoi element of $V^k(A)$ in the interior of $reg(H \cup \{c\})$ gets deleted from $V^{k+1}(A)$ (unless colored red, see below) and all subregions of $reg(H \cup \{c\})$ get merged into a new $(k+1)$-order region of $V^{k+1}(A)$. It remains to determine whether $H \cup \{c\}$ forms a cut for the biconnected component B such that $c \in B$. There are two cases:

1. If $reg(H \cup \{c\})$ is incident to an already red Voronoi region $reg(R)$ of $V^k(A)$ such that $R \subset H \cup \{c\}$ then clearly $H \cup \{c\}$ forms a cut. Then $reg(H \cup \{c\})$ is colored red and $reg(H \cup \{c\})$ gets merged within $reg(R)$. Since no portion of

$reg(H \cup \{c\})$ is already red in $V^k(A)$ it is not hard to see that the portion of $V^k(A)$ in the interior of $reg(H \cup \{c\})$ is in fact the corresponding portion of the farthest Voronoi diagram of R, $V_f(R)$. We say that that the region of the cut R *expands* into $reg(H \cup \{c\})$ keeping R as the sole owner of the expanded region. The portion of $V^k(A)$ in the interior of $reg(H \cup \{c\})$ remains as a finer subdivision of the expanded region.

2. Otherwise we need to check whether $H \cup \{c\}$ forms a new cut (see below), i.e., whether $reg(H \cup \{c\})$ becomes a new red region of $V^{k+1}(A)$. If $H \cup \{c\}$ is indeed a new cut of biconnected component B, let $C = B \cap (H \cup \{c\})$; C is assigned as the owner of $reg(H \cup \{c\})$, which is now denoted simply as $reg(C)$, and it is colored red. The interior of $reg(C)$ gets partitioned into finer subregions by $V_f(C)$, the farthest Voronoi diagram of C, as given by the portion of $V^k(A)$ in the interior of $reg(C)$. It is not hard to see that no information is lost by assigning C as the owner of $reg(H \cup \{c\})$ as no core element in $(H \cup \{c\}) - C$ can be the farthest one among elements of $H \cup \{c\}$ for any point $t \in reg(H \cup \{c\})$. In fact any element of H that is not represented in the (complete) farthest Voronoi diagram of C, can be excluded from C.

The following Lemma can be derived using the properties of standard higher order Voronoi diagrams of points (see [11]).

Lemma 2. *The opens Voronoi diagram on layer A has size $O(k(n-k))$, where n denotes the number of polygonal edges on layer A, and k is the maximum number of iterations performed in the construction of the diagram until all regions are colored red. The number k depends on the connectivity of $G(N, A)$.*

A simple (almost brute force) algorithm to determine new cuts of $V^{k+1}(A)$ is as follows. Let $reg(H)$ be a non-red region of $V^k(A)$ and let B be a biconnected component associated with set H. That is, $B \cap H \neq \emptyset$ and there is a Voronoi edge bounding $reg(H)$ induced by core elements b, h such that $b \in B - H$ and $h \in H$. Remove the elements of H from B and determine new non-trivial bridges, articulation points and biconnected components of $B - H$. Clearly $H \cup \{c\}$ is a new cut of B if and only if c is a new non-trivial bridge or articulation point of $B - H$. It is now easy to determine any new cut associated with the non-red Voronoi edges or vertices bordering $reg(H)$ in $V^k(A)$. Note that any new cut of $V^{k+1}(A)$ corresponds to at least one Voronoi element of $V^k(A)$.

To determine the new cuts of $V^2(A)$ (i.e., generators of order 2) a much faster algorithm could be obtained by partitioning biconnected components of $G(N, A)$ into *triconnected components* (see [16]). However this is not easily generalizable to $k > 2$. Many biconnected components in actual VLSI designs are expected to have low connectivity to the extent of being simple cycles. For simple cycles the problem is easy and can be answered using a simple coloring scheme on the DFS tree of the corresponding biconnected component.

The time complexity of the entire algorithm is described in the following lemma.

Lemma 3. *The Voronoi diagram for opens on a VLSI layer A can be computed in time $O(k^2 n \log n)$ plus a total $O(k^2 n^2)$ time to determine cuts associated with higher order generators, where k is the maximum number of iterations performed and n is the complexity of layer A. In case of biconnected components forming simple cycles or if the maximum number of iterations is bounded by $k = 2$ the bound simplifies to $O(n \log n)$.*

6 A Hausdorff Voronoi Diagram for Opens

Once the set of cuts $\mathcal{C}$ claiming a region in the opens Voronoi diagram on layer A have been identified, the opens Voronoi diagram can be interpreted as the Hausdorff Voronoi diagram of $\mathcal{C}$. Given a cut C and a point t, the Hausdorff distance between t and C simplifies to the maximum (weighted) distance of t from any core element in C, i.e., $d_h(t, C) = \max\{d_w(t, c), c \in C\}$. The Hausdorff Voronoi diagram of a set of cuts S is the Voronoi diagram of S under the Hausdorff distance, where the Hausdorff Voronoi region of a cut C is $reg(C) = \{t \mid d_h(t, C) \leq d_h(x, C_j), C_j \in S\}$ (for the definition of an ordinary Hausdorff Voronoi diagram see e.g [17,3]). Assuming that some superset of cuts $S \supseteq \mathcal{C}$ can be identified, the Hausdorff Voronoi diagram of S provides an alternative definition for the opens Voronoi diagram on a layer A. This observation can help reduce the number of iterations in computing the final opens Voronoi diagram and speed up the construction in practice. Namely, once a sufficient set of cuts $\mathcal{C}'$ has been identified the iteration can stop and the Hausdorff Voronoi diagram of $\mathcal{C}'$ can be directly computed in the non-red portion of the current $V^k(A)$. Furthermore, one can localize the higher order Voronoi diagram computation by applying the iterative process to identify cuts to each biconnected component independently. Computing the Hausdorff Voronoi diagram of cuts for all biconnected components union the 1st order generators provides the opens Voronoi diagram. Practical limits on the number of iterations for each biconnected component can be easily imposed to gain speed with only minimal potential loss in accuracy (if any). For a plane sweep algorithm to compute the Hausdorff Voronoi diagram for clusters of points see [3].

Often it is desirable to compute one critical area value combining interconnect opens on layer A and *via-blocks* on the neighboring via layers into a single estimate of critical area for *missing material defects*. A via-block is a defect entirely destroying a connection (a via or cluster of vias) between neighboring conducting layers. The problem of computing critical area for via-blocks reduces to computing a Hausdorff Voronoi digram of polygons representing clusters of redundant vias (see [2,3]). To compute the combined Voronoi diagram for missing material defects we simply need to substitute $V(A)$, with $V_h(A')$, the Hausdorff Voronoi diagram of all core elements on layer A union clusters of vias on the neighboring via layers. Voronoi regions of via-clusters represent via-blocks and are always colored red. $V_h(A')$ can be computed by plane sweep by adapting the plane sweep construction of [3]. The iterative process to compute new cuts and the final Voronoi diagram for missing material defects remains similar.

Acknowledgments

The author would like to thank Dr. L. F. Heng of IBM T.J. Watson Research center for helpful discussions on the definition of a net graph. The IBM Voronoi EDA group including R. Allen, S.C. Braasch, J.D. Hibbeler and M.Y. Tan are greatly acknowledged for jointly developing, supporting, and expanding the Voronoi Critical Area Analysis Tool.

References

1. Papadopoulou, E., Lee, D.T.: Critical area computation via Voronoi diagrams. IEEE Transactions on Computer-Aided Design 18(4), 463–474 (1999)
2. Papadopoulou, E.: Critical area computation for missing material defects in VLSI circuits. IEEE Transactions on Computer-Aided Design 20(5), 583–597 (2001)
3. Papadopoulou, E.: The Hausdorff Voronoi diagram of point clusters in the plane. Algorithmica 40, 63–82 (2004)
4. Stapper, C.H., Rosner, R.J.: Integrated circuit yield management and yield analysis: Development and implementation. IEEE Trans. Semiconductor Manufacturing 8(2), 95–102 (1995)
5. Stapper, C.H.: Modeling of defects in integrated circuit photolithographic patterns. IBM J. Res. Develop. 28(4), 461–475 (1984)
6. Ferris-Prabhu, A.: Defect size variations and their effect on the critical area of VLSI devices. IEEE J. of Solid State Circuits 20(4), 878–880 (1985)
7. Kahng, A.B., Liu, B., Mandoiu, I.I.: Non-tree routing for reliability and yield improvement. IEEE Trans. on Comp. Aided Design of Integrated Circuits and Systems 23(1), 148–156 (2004)
8. Aurenhammer, F., Drysdale, R., Krasser, H.: Farthest line segment Voronoi diagrams. Information Processing Letters (100), 220–225 (2006)
9. Papadopoulou, E., Lee, D.: The l_∞ Voronoi diagram of segments and VLSI applications. International Journal of Computational Geometry and Applications 11(5), 503–528 (2001)
10. Maynard, D.N., Hibbeler, J.D.: Measurement and reduction of critical area using Voronoi diagrams. In: Advanced Semiconductor Manufacturing IEEE Conference and Workshop (2005)
11. Lee, D.T.: On k-nearest neighbor Voronoi diagrams in the plane. IEEE Trans. Comput. C-31(6), 478–487 (1982)
12. Tarjan, R.: Depth-first search and linear graph algorithms. SIAM Journal on Computing 1, 146–160 (1972)
13. Hopcroft, J., Tarjan, R.: Efficient algorithms for graph manipulation. Comm. ACM 16(6), 372–378 (1973)
14. Aurenhammer, F.: A new duality result concerning Voronoi diagrams. Discrete and Cpomputational Geometry 5, 243–254 (1990)
15. Chazelle, B., Edelsbrunner, H.: An improved algorithm for constructing kth order Voronoi diagrams. IEEE Transactions on Computers C-36, 1349–1354 (1987)
16. Hopcroft, J., Tarjan, R.: Dividing a graph into triconnected components. SIAM Journal on Computing 2, 135–158 (1973)
17. Aurenhammer, F., Klein, R.: Voronoi diagrams. In: Sack, J., Urrutia, G. (eds.) Handbook of Computational Geometry, pp. 201–290. Elsevier, Amsterdam (2000)

Distributed Relationship Schemes for Trees

Cyril Gavoille* and Arnaud Labourel

LaBRI, Université de Bordeaux, France
{gavoille,labourel}@labri.fr

Abstract. We consider a distributed representation scheme for trees, supporting some special relationships between nodes at small distance. For instance, we show that for a tree T and an integer k we can assign local information on nodes such that we can decide for any two nodes u and v if the distance between u and v is at most k and if so, compute it only using the local information assigned. For trees with n nodes, the local information assigned by our scheme are binary labels of $\log n + O(k \log(k \log(n/k)))$ bits, improving the results of Alstrup, Bille, and Rauhe (SODA '03).

Keywords: distributed data-structures, ancestry, tree, distance.

1 Introduction

A *distributed representation scheme* is a scheme maintaining global information on a network using local data structures (or labels) assigned to nodes of the network. Such schemes play an important role in the fields of distributed computing. Their goal is to locally store some useful information about the network and make it conveniently accessible. For instance, *implicit representation* of networks is a distributed representation scheme that supports adjacency queries, i.e., adjacency between two nodes can be determined only by examining the local information stored by the two nodes. So, the network can be manipulated by keeping only its labels in memory, any other global information on the graph (like its matrix) can be removed. The goal is to minimize the maximum length of a label associated with a vertex while keeping fast queries.

Distributed representation is widely used in distributed computing, e.g. in [14,18]. Kannan, Naor and Rudich [17] investigated in particular implicit representation for several families of graphs, including trees with labels of $2 \log n$ bits[1] where n is the number of nodes of the tree. Actually, parent and ancestry queries for rooted trees can be done with $1.5 \log n + O(\log \log n)$ bit labels [2], and this has been improved to $\log n + O(\sqrt{\log n})$ [1]. If we insist only on implicit representation of trees, the label length can be reduced. Chung [9] improved in a non-trivial way to $\log n + \log \log n + O(1)$ bits, and further improved to $\log n + O(\log^* n)$ bits[2] by Alstrup et al. [6].

* Both authors are supported by the project CÉPAGE of INRIA Futur, and the ANR-projects GEOCOMP and GRAAL.

[1] The log function denotes the logarithm in base two.

[2] $\log^* n$ denotes the number of times log should be iterated to get a constant.

T. Tokuyama (Ed.): ISAAC 2007, LNCS 4835, pp. 728–738, 2007.

1.1 Related Works

Motivated by applications in XML search engines, and distributed applications as peer-to-peer networks or network routing, several queries on distributed data-structures have been investigated recently. In this framework, distributed data-structure can be seen as a label assigned to each node such that queries can be answered by inspecting the labels only, without any other source of information. For instance, address-based routing in trees [11,19,21] or in a specific class of networks [8,10], distance queries for cycles [20], for interval and permutation graphs [7,13], or for hyperbolic graphs [3,12], ancestry in rooted trees [1], etc. have optimal $O(\log n)$-bit distributed data-structures.

In this paper, we consider labeling scheme for various relationships between nodes of small distance in trees. For instance, we construct a simple distributed representation scheme supporting parent and sibling queries with labels of size $\log n + 2\log\log n + O(1)$, the best bound was $\log n + 5\log\log n + O(1)$ [5]. The current lower bound on the label length for schemes supporting sibling queries in trees is $\log n + \log\log n + O(1)$ [4].

We stress that improving the second order term on the label length complexity is not only of theoretical interest. It does matter in practice because, as mentioned in [2], engines for indexing XML files use such schemes for huge database in which each item is associated with a label. Therefore decreasing by one byte the length of the labels results a save of gigabytes of main memory for large database. Moreover and interestingly, every distributed representation can be interpreted as a universal matrix [20]: row and column indexes of the matrix correspond to all the labels of the scheme, and the value of entry (i, j) in the matrix is the answer of the query applied to two nodes labeled respectively i and j by the scheme. In the case of implicit representation for instance, the universal matrix is Boolean and corresponds to the adjacency matrix of a graph, called *induced-universal graph*, having the property of containing all graphs of the family as induced subgraph. So, proving a complexity of $f(n)$ for label length transfers to the existence of universal matrix of dimension $O(2^{f(n)})$. So improving the second order term in the label length actually improves the complexity of the matrix dimension. For concreteness, the difference between the $2\log n$-bit labeling scheme of [17] and the $\log n + \log\log n + O(1)$ scheme of [9] corresponds to an improvement from $O(n^2)$ to $O(n\log n)$ on the size of the smallest induced-universal graphs containing all n-node trees as induced subgraphs.

1.2 Our Contribution

In a rooted tree, two nodes u and v with nearest common ancestor z are (k_1, k_2)-*related* if the distance from u to z is k_1 and the distance from v to z is k_2. For any integer k, a *k-relationship scheme* is a distributed representation scheme that supports tests for whether u and v are (k_1, k_2)-related for all nodes u and v, and all integers $k_1, k_2 \leqslant k$.

For instance, a 1-relationship scheme supports tests for whether two nodes are $(0, 0)$, $(1, 0)$, $(0, 1)$ or $(1, 1)$-related, and so supporting parent and sibling queries.

Let us observe that a k-relationship scheme supports also distance queries for nodes at distance at most k.

In this paper we propose a k-relationship scheme for trees of n nodes with labels of size $\log n + O(k \log(k \log(n/k)))$ bits. This improves the scheme presented in [4,5] that uses labels of $\log n + O(k^2 \log(k \log n))$ bits. Our scheme is simple and easy to implement, and we show[3] that the time to preprocess the tree and for constructing the n labels is $O(kn + n \log n)$. Our scheme also implies that distance between nodes at distance $o(\log n / \log\log n)$ can be determined with labels of $\log n + o(\log n)$ bits. In contrast, it has been proved in [15] that labels of $\Omega(\log^2 n)$ bits are required for arbitrary distance queries in trees.

A k-relationship scheme has query time complexity t if one can check whether a pair of nodes is (k_1, k_2)-related or not, in time at most t, for any pair of nodes and all integers $k_1, k_2 \leqslant k$.

Our result is summarized in the following statement:

Theorem 1. *The family of n-node rooted trees enjoys a k-relationship scheme with labels of $\log n + O(k \log(k \log(n/k)))$ bits with constant query time.*

For $k = 1$, we show that labels are of length $\log n + 2 \log\log n + O(1)$, improving the $\log n + 5 \log\log n + O(1)$ scheme of [5]. Moreover, for $k = o(\log n / \log\log n)$, we derive directly from Theorem 1 that:

Corollary 1. *The family of n-node trees enjoys a distributed representation supporting distance with labels of $\log n + o(\log n)$ bits for nodes at distance $o(\log n / \log\log n)$.*

In Section 2, we present the k-relationship scheme, and we conclude in Section 3 with some open problems.

2 A Relationship Scheme

2.1 Preliminaries

The basic idea of our scheme is to store into the label of each node u, some identifiers for u and for its k closest ancestors. Indeed, to test if u and v are (k_1, k_2)-related, it suffices to test if the ancestor at distance k_1 of u is equal to the ancestor at distance k_2 of v, and the ancestor at distance $k_1 - 1$ of u differs from the ancestor at distance $k_2 - 1$ of v. A naive implementation would lead to $O(k \log n)$-bit labels for arbitrary identifiers. We can significantly decrease this complexity with a better choice of the identifiers exploiting correlations between nodes.

Let T be a rooted tree of n nodes. We define the *k-ancestry* of a node u as the set of ancestors of u at distance at most k, u included. A *branch* of T is a path leading from the root to a leaf of T. We denote by $G[X]$ the subgraph induced by the set of nodes X.

The main notion we introduce below is illustrated on Fig. 2.1.

[3] Due to space limitation the time complexity of the scheme is detailed in the full version only.

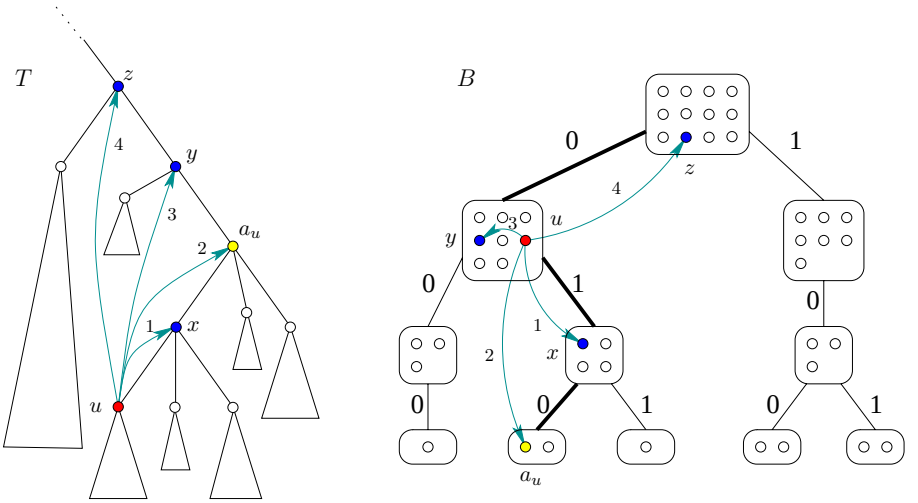


Fig. 1. An example of a k-ancestor decomposition B of an input tree T where the 4-ancestry of u, the set $A_u = \{u, x, a_u, y, z\}$, lies on the branch "010" of B

Definition 1. *A* k-ancestor decomposition *of a rooted tree* T *is a rooted binary tree* B *where nodes, called* parts*, form a partition of* $V(T)$ *such that, for any* k*-ancestry* A *of* T*, the set of parts containing a node of* A *is contained in a branch of* B*.*

2.2 Finding an Nice Ancestor Decomposition

This paragraph is devoted to the proof of the following key lemma for our result.

Lemma 1. *Every* n*-node tree* T *has a* k*-ancestor decomposition such that every part of depth* h *contains at most* $(k+1) \cdot (\log(2n/(k+1)) - h)$ *nodes of* T*.*

Before the formal proof of this result, let us give an overview. The idea is to construct from T a graph G, called hereafter *k-augmentation of T*, obtained by adding an edge between every u and its proper ancestors at distance $\leqslant k$. We then observe that every subgraph H of G has a subset of $k+1$ nodes, called *half-separator*, whose removal leaves H in connected components with less than $|V(H)|/2$ nodes. The root of the willing decomposition B is constructed by finding iteratively $O(\log(n/k))$ half-separators in G, and by grouping all the resulting connected components in two sets V_1, V_2, each with less than $n/2$ nodes. By this way we can guarantee that there are no edges of G between V_1 and V_2 since these two sets are separated by an union of separators in G. The tree B is completed by performing similarly and recursively the subgraphs of $G[V_1]$ and $G[V_2]$. Eventually, B is a k-ancestor decomposition since we observe that:

1. k-ancestries of T induce cliques in G; and
2. all edges of G belong to the same branch in B.

Let us now formalize the proof, and let us first show that:

Property 1. Every induced subgraph of the k-augmentation of T has a half-separator of size at most $k+1$ that is a clique.

Proof. It is known [16] that every *chordal* graph G, that is a graph without any induced cycle of length greater than three, has a half-separator of size at most the maximum clique size of G. Moreover, such a half-separator can be computed in linear time, i.e., $O(|V(G)| + |E(G)|)$ time.

So Property 1 is derived from the fact that every induced subgraph of a chordal graph is a chordal graph as well, and that the k-augmentation of T is a chordal graph of maximum clique size $k+1$.

Let G be the k-augmentation of T. Assume G has a clique K of size $> k+1$. Observe that in G there is no edge between unrelated nodes of T. (Two nodes of a rooted tree are said *related* if one is the ancestor of the other. They are *unrelated* otherwise.) In other words, all the nodes of K belong to the same branch of T. It follows that the distance in T between the closest node (from the root of T), say u, and the farthest node, say v, of K is $\geqslant k+1$. Because u and v are related and are at distance $> k$ in T, they are not adjacent in G: a contradiction with the fact that u and v belong to a clique of G.

Assume now G has a induced cycle C length > 3. Since G has no edges between unrelated nodes, there must exist three consecutive nodes in C, say u, v, w, such that u is ancestor of v, and w is ancestor of v (otherwise C would be a path). Either w is ancestor of u, or u is ancestor of w. In both cases, u and w are related and at distance $\leqslant k$ in T, so there are adjacent in G: a contradiction with the fact that C is an induced cycle of length > 3.

We have therefore proved that G is chordal and of maximum clique size $\leqslant k+1$. □

In the following let us denote by G the k-augmentation of T, and let us denote by Half-Separator(H) a function that returns a half-separator for a subgraph H satisfying Property 1, that is a separator whose size is at most the maximum clique size of H.

We now restrict our attention on specific half-separators. A half-separator S of a graph H is said *binary* if the connected components of $H \setminus S$ can be partitioned in at most two sets, each with a total number of nodes at most $|V(H)|/2$.

We consider the following procedure (Algorithm 1) that given a graph satisfying Property 1 returns a binary half-separator R and the resulting partition (V_1, V_2) for the nodes of $H \setminus R$.

To construct the k-ancestor decomposition B for T we apply Algorithm 1 on G, and we select R as the root of B. The tree B is then completed by applying recursively Algorithm 1 on $G[V_1]$ and $G[V_2]$, and linking to R the resulting decompositions if they are non-empty. Such a recursive approach is possible because Property 1 holds for any induced subgraph of G, so in particular for $G[V_1]$ and $G[V_2]$.

Eventually B is a k-ancestor decomposition of T because every k-ancestry of T induces a clique in G, and no edge connects $G[V_1]$ to $G[V_2]$ since R is a

Algorithm 1: BINARY HALF-SEPARATOR

Input : a subgraph G satisfying Property 1

Output: a binary half-separator R and the associated node partition (V_1, V_2)

$H := G$; $V_1 := V_2 := R := \varnothing$
while $|V(H)| > k+1$ **do**
  $S :=$ HALF-SEPARATOR(H); $H := H \setminus S$; $R := R \cup S$
  forall *connected component* C *of* H *except the largest* **do**
    $H := H \setminus C$;
    if $|V_1| > |V_2|$ **then** $V_2 := V_2 \cup V(C)$ **else** $V_1 := V_1 \cup V(C)$
$R := R \cup H$

half-separator. So, every k-ancestry of T belongs to parts contained in a same branch of B.

It remains to estimate the size of R, V_1, and V_2 returned by Algorithm 1.

At each iteration of the while-main the size of H is divided by at least two since all the resulting components of $H \setminus S$ are removed from H, and the remaining largest component is of size at most $|V(H)|/2$. So, there is at most $\log(n/(k+1))$ iterations where $n = |V(G)|$.

At each step the size of R increases by at most $k+1$ vertices, and the final step add at most $k+1$ vertices to R. Therefore, $|R| \leqslant (k+1) \cdot (\log(n/(k+1)) + 1) = (k+1) \cdot (\log(2n/(k+1)) - h)$ with $h = 0$.

In order to insure the correctness of the decomposition, we need to show the following loop invariant (P).

$$(P): \quad |V_1|, |V_2| \leqslant n/2 \ \text{ and } \ V(H) \cup V_1 \cup V_2 \cup R = V(G)$$

It is straightforward to verify that (P) is true at the beginning of the main loop. Let us show that (P) remains true at the end of the nested loop. The loop invariant clearly remains true after computation of half-separator and statement $H := H \setminus S$ and $R := R \cup S$. Assume w.l.o.g. that $|V_1| \geqslant |V_2|$. We have $V(H) \cup V_1 \cup V_2 \subseteq V(G)$ since (P) is true. We obtain the following relation for the size of the sets.

$$\begin{aligned}
|V(H)| + |V_1| + |V_2| &\leqslant n \\
|V(H)| &\leqslant n - |V_1| - |V_2| \ \leqslant \ n - 2|V_2| \quad (|V_1| \geqslant |V_2|) \\
|V(H)|/2 &\leqslant n/2 - |V_2| \\
|V(C)| &\leqslant n/2 - |V_2| \quad \text{(there is yet another larger component in } H) \\
|V(C)| + |V_2| &\leqslant n/2
\end{aligned}$$

Property (P) remains true at the end of the nested loop. Property (P) is a loop invariant and so is true at the end of the main loop. We have that $|V_1|, |V_2| \leqslant n/2$. Moreover, there is no edge linking vertices of V_1 to vertices of V_2 since what we add to V_1 or V_2 is a full connected component of H.

Since the size of V_1 and V_2 are $\leqslant n/2$, by induction, the size of the root part in the decomposition of $G[V_1]$ and $G[V_2]$ (so parts of depth $h = 1$ in B) would

be at most $(k+1)\cdot(\log(2(n/2)/(k+1))) = (k+1)\cdot(\log(2n/(k+1))-1)$. And more generally, for an arbitrary depth $h \geqslant 0$, the parts of depth h are of size at most $(k+1)\cdot(\log(2(n/2^h)/(k+1)))) = (k+1)\cdot(\log(2n/(k+1))-h)$ as claimed.

This completes the proof of Lemma 1.

2.3 The Labels

Let X be a part of B at depth h. We denote by $\mathrm{path}(X)$ the binary word of length h defining the unique path from the root of T to X. We associated with each $u \in X$ its *rank*, a unique integer $\mathrm{rank}(u) \in [0, |X|)$, and its *position*, defined by the pair $\mathrm{pos}(u) = (h, \mathrm{rank}(u))$.

The *apex* of a k-ancestry A is the node $a \in A$ with the deepest part and, if equality, with the largest rank (see the yellow node of Fig. 2.1). Observe that the positions are relative to a branch of B: every pair of nodes whose parts are on the same branch have distinct positions, and thus the parts of any two nodes having the same positions cannot be related.

Let u be a node of T, let A_u be its k-ancestry, a_u the apex of A_u, and B_{a_u} the part in B that contains a_u. The label of u is defined by the following quadruple:

$$\mathrm{label}(u) = (\mathrm{path}(B_{a_u}), \mathrm{rank}(a_u), d_{a_u}, P_u)$$

where:

- d_{a_u} is the distance in T from u to a_u; and
- $P_u = \{\mathrm{pos}(v) \mid v \in A_u, v \neq a_u\}$.

In order to optimize the query time, we assume that P_u is implemented as an array ordered by distances from u. For concreteness, the label of u in the example of Fig. 2.1 is[4]:

$$\mathrm{label}(u) = (\text{"}010\text{"}, 0, 2, \{(1,5),(2,0),(1,3),(0,9)\}) \ .$$

Let $\mathrm{pos}(A_u) = \{\mathrm{pos}(v) \mid v \in A_u\}$. It is not difficult to see that any k-ancestry A_u is uniquely defined by the pair $(\mathrm{pos}(A_u), \mathrm{path}(B_{a_u}))$, i.e., the set of its positions and the path leading to its apex. Indeed, as said previously, nodes lying on a same branch of B have pairwise distinct positions, and nodes lying on different branches can be identified from the path of their apices (that must therefore differ). The set $\mathrm{pos}(A_u)$ is not a field of $\mathrm{label}(u)$, P_u misses $\mathrm{pos}(a_u)$. However, it can be computed since $\mathrm{pos}(a_u) = (|\mathrm{path}(B_{a_u})|, \mathrm{rank}(a_u))$.

We first bound the length of the labels. Then we will explain how to solve a (k_1, k_2)-relation query, and we will detail an efficient implementation.

Lemma 2. *The label length is* $\log n + O(k \log(k \log(n/k)))$. *For* $k = 1$, *the label length is* $\log n + 2\log\log n + O(1)$.

[4] Ranks of nodes in each part are ordered by rows from left to right and then from top to bottom rows.

Proof. Let $M = \lfloor (k+1)\log(2n/(k+1)) \rfloor$. By Lemma 1, the parts of B have at most M nodes. Consider label(u), and let $h = |\text{path}(B_{a_u})|$ be the depth of B_{a_u}.

The binary word path(B_{a_u}) is of length exactly h. We have rank$(a_u) \in [0, M)$, so $\log M + O(1)$ bits suffice. The value $d_{a_u} \in [0, k]$, so $O(\log k)$ bits suffice. Each position $(h', r') \in P_u$ can be stored with $\log h + \log M + O(1)$ bits since $h' \leqslant h$ and $r' < M$. Moreover, $|P_u| = k$, so $k \cdot (\log h + \log M) + O(k)$ bits suffices for P_u.

Overall, given h, the length of label(u) is at most:

$$|\text{label}(u)| \ \leqslant \ h + \log M + k \cdot (\log h + \log M) + O(k) \ .$$

We will now upper bound the length of each term by a function depending only on the parameters of the problem (here n and k), and not on parameters depending on a tree or a particular node, like h. Therefore, each of the four fields of label(u) can be coded by a binary string of predefined length, and do not require extra field separators.

By Lemma 1, the size of the parts in B that are at depth $\log(n/(k+1))$ are of size at most $(k+1) \cdot (\log(2n/(k+1)) - \log(n/(k+1))) \leqslant k+1$. From the while-condition in Algorithm 1, if $|V(H)| \leqslant k+1$, then the input graph is not separated at all, implying that the part is actually a leaf of B. Therefore B has depth at most $h_0 \leqslant \log(n/(k+1)) < \log(n/k)$. So bounding $h \leqslant h_0$ we obtain:

$$\begin{aligned}
|\text{label}(u)| &\leqslant h_0 + \log M + k \cdot (\log h_0 + \log M) + O(k) \\
&\leqslant h_0 + k \log h_0 + (k+1) \log M + O(k) \\
&\leqslant \log(n/k) + k \log\log(n/k) + (k+1) \log((k+1)\log(2n/(k+1))) + O(k) \\
&\leqslant \log(n/k) + (2k+1) \log\log(n/k) + O(k \log k) \\
&\leqslant \log(n/k) + O(k) \cdot (\log\log(n/k) + \log k) \\
&\leqslant \log n + O(k \log(k \log(n/k))) \ .
\end{aligned}$$

For $k = 1$, the above formula gives $\log n + 3\log\log n + O(1)$ from the above 4th equation. We can slightly improved the above analysis by observing that the two first fields of label(u), namely path(B_{a_u}) and rank(a_u), can be jointly encoded with $\log n + O(1)$ bits instead of $h_0 + \log M \sim \log(n/k) + \log\log(n/k)$ bits.

We remark that for $k = 1$, the k-augmentation G of T is T itself. As a consequence, the size of the half-separator in Property 1 can be reduced since it is well-known that every forest has a single node that halves the forest, rather than a clique separator of size $k+1 = 2$. It follows that in such 1-ancestor decomposition of T the parts of depth h contain at most $\alpha = \log n - h + O(1)$ nodes instead of $(k+1)(\log(2n/(k+1)) - h) = 2(\log n - h)$ as claimed in Lemma 1. A direct consequence is that the two first terms of label(u) can be coded jointly by a string W of $\log n + O(1)$ bits as follows: W is composed of path(B_{a_u}) of length h concatenated to the word $1 \circ 0^{\text{rank}(a_u)}$, i.e., the unary representation of rank(a_u) preceded with a 1. The length of this word is $h + 1 + \alpha = \log n + O(1)$, and clearly the two fields can be extracted by seeking the least significant bit of W. The two remaining fields of label(u) still have a length bounded by

$2k \log\log(n/k) + O(k \log k)$. Overall, the label length of label(u), for $k = 1$, is at most $\log n + 2 \log\log n + O(1)$.

This completes the proof of Lemma 2. □

Actually a finer analysis shows that the label length for $k = 1$ is no more than $\log n + 2 \log\log n + 2$ for every $n \geqslant 16$.

2.4 Relationship Testing

Let u, v be two nodes of T with k-ancestry A_u and A_v respectively. Consider any pair (k_1, k_2) of integers $\leqslant k$. Recall that to check whether (u, v) is (k_1, k_2)-related or not it suffices to check if the nearest common ancestor between u and v (which is a node $z \in A_u \cap A_v$) is at distance k_1 from u and k_2 from v.

Observing that there is exactly one ancestor $z_u \in A_u$ at distance k_1 from u, and one ancestor $z_v \in A_v$ at distance k_2 from v, it suffices to check that $z_u = z_v$, and that z_u is the least common ancestor, that is there is no $z \in A_u \cap A_v$ at distance $k_1 - 1$ from u and $k_2 - 1$ from v.

Let label(u) = $(\text{path}(B_{a_u}), \text{rank}(a_u), d_{a_u}, P_u)$ and label(v)= $(\text{path}(B_{a_v}), \text{rank}(a_v), d_{a_v}, P_v)$. We denote by $\text{path}(X)[0..h]$ the prefix of length h of[5] $\text{path}(X)$.

The following lemma tell us how to check that $z \in A_u \cap A_v$ from the position of z and the labels of u and v. Recall that positions are relative to the branches of B, so a given position (h, r) may correspond to different nodes of T. Unfortunately, we cannot simply check that $\text{pos}(z) \in \text{pos}(A_u) \cap \text{pos}(A_v)$.

Property 2. Assume $z \in A_u$, and $\text{pos}(z) = (h, r)$. Then, $z \in A_v$ if and only if $\text{pos}(z) \in \text{pos}(A_v)$ and $\text{path}(B_{a_u})[0..h] = \text{path}(B_{a_v})[0..h]$.

Proof. Let $z \in A_u$, and let B_z be its part in the decomposition B. We have that $\text{path}(B_z)$ is a prefix of $\text{path}(B_{a_u})$ since $z \in A_u$. If the depth of z is h, then $\text{path}(B_z) = \text{path}(B_{a_u})[0..h]$. Similarly, $z \in A_v$ implies that $\text{path}(B_z) = \text{path}(B_{a_v})[0..h]$. Obviously, $z \in A_v$ implies $\text{pos}(z) \in \text{pos}(A_v)$. Therefore, we have shown that $z \in A_v$ implies $\text{pos}(z) \in \text{pos}(A_v)$ and $\text{path}(B_{a_u})[0..h] = \text{path}(B_{a_v})[0..h] = \text{path}(B_z)$.

Conversely, if $\text{pos}(z) = (h, r) \in \text{pos}(A_v)$ and $\text{path}(B_{a_u})[0..h] = \text{path}(B_{a_v})[0..h]$, then, since $z \in A_u$, the part of z, B_z is given by $\text{path}(B_z) = \text{path}(B_{a_u})[0..h] = \text{path}(B_{a_v})[0..h]$. It follows that the position (h, r) corresponds to a node of B_z which is moreover in A_v. In B_z, there is a unique node whose rank is r: node z. So $z \in A_v$. □

For every distance $i \in \{0, \ldots, k\}$, let us denote by $\text{pos}(A_u)[i]$ the position of the ith ancestor of u, i.e., at distance i from u. Note that $\text{pos}(A_u)[i]$ can be extracted from label(u) in constant time as follows (assuming that P_u is ordered according to the distance from u):

Extract $\text{pos}(A_u)[i]$ (given label(u)):

[5] Such prefix can be extracted (shift) in constant time in the word RAM model, because $\text{path}(X)$ is a binary word of length $\leqslant \log n$.

1. If $i = d_{a_u}$, then return $\mathrm{pos}(a_u)$, i.e., $(|\mathrm{path}(B_{a_u})|, \mathrm{rank}(a_u))$.
2. If $i > d_{a_u}$, then $i = i - 1$.
3. Return $P_u[i]$.

According to Lemma 2, to check whether $z \in A_u \cap A_v$ we have to check that $\mathrm{pos}(z) = (h, r) \in \mathrm{pos}(A_u) \cap \mathrm{pos}(A_v)$ and that the two prefixes of length h corresponds. This leads to the following test procedure:

FINAL TEST (whether (u, v) is (k_1, k_2)-related given label(u) and label(v)):

1. Extract $(h_u, r_u) = \mathrm{pos}(A_u)[k_1]$ and $(h_v, r_v) = \mathrm{pos}(A_v)[k_2]$.
2. If $(h_u, r_u) \neq (h_v, r_v)$, then return FALSE.
3. If $\mathrm{path}(B_{a_u})[0..h_u] \neq \mathrm{path}(B_{a_v})[0..h_v]$, then return FALSE.
4. If $k_1 = 0$ or $k_2 = 0$, then return TRUE.
5. Extract $(h'_u, r'_u) = \mathrm{pos}(A_u)[k_1 - 1]$ and $(h'_v, r'_v) = \mathrm{pos}(A_v)[k_2 - 1]$.
6. If $(h'_u, r'_u) \neq (h'_v, r'_v)$, then return TRUE.
7. If $\mathrm{path}(B_{a_u})[0..h'_u] \neq \mathrm{path}(B_{a_v})[0..h'_v]$, then return TRUE.
8. Return FALSE.

Lemma 3. *Any k-relationship can be tested in constant time.*

Proof. The above procedure clearly takes a constant time, and its validity is derived from Property 2. □

Combining Lemma 1, Lemma 2, and Lemma 3, we have proved Theorem 1.

3 Conclusion and Further Works

We have constructed in this paper a k-relationship scheme for n-node trees with $\log n + O(k \log(k \log(n/k)))$ bit labels. This scheme implies that distances in trees can be computed as well with labels of $\log n + o(\log n)$ bits if the distance is $o(\log n / \log\log n)$. We leave open the following two questions:

- Design a distance labeling scheme for trees with $\log n + o(\log n)$ bit labels and for larger distances, say for distances up to $\log n$.
- Design a distance labeling scheme for small distances for bounded treewidth graphs.

References

1. Abiteboul, S., Alstrup, S., Kaplan, H., Milo, T., Rauhe, T.: Compact labeling schemes for ancestor queries. SIAM Journal on Computing 35, 1295–1309 (2006)
2. Abiteboul, S., Kaplan, H., Milo, T.: Compact labeling schemes for ancestor queries. In: 12^{th} Symposium on Discrete Algorithms (SODA), pp. 547–556 (January 2001)
3. Abraham, I., Balakrishnan, M., Kuhn, F., Malkhi, D., Talwar, K., Ramasubramanian, V.: Reconstructing approximate tree metrics. In: 26^{th} Annual ACM Symposium on Principles of Distributed Computing (PODC), pp. 43–52. ACM Press, New York (2007)

4. Alstrup, S., Bille, P., Rauhe, T.: Labeling schemes for small distances in trees. In: 14^{th} Symposium on Discrete Algorithms (SODA), ACM-SIAM, pp. 689–698 (2003)
5. Alstrup, S., Bille, P., Rauhe, T.: Labeling schemes for small distances in trees. SIAM Journal on Discrete Mathematics 19, 448–462 (2005)
6. Alstrup, S., Rauhe, T.: Small induced-universal graphs and compact implicit graph representations. In: 43^{rd} Annual IEEE Symposium on Foundations of Computer Science (FOCS), pp. 53–62. IEEE Computer Society Press, Los Alamitos (2002)
7. Bazzaro, F., Gavoille, C.: Localized and compact data-structure for comparability graphs. In: Deng, X., Du, D.-Z. (eds.) ISAAC 2005. LNCS, vol. 3827, pp. 1122–1131. Springer, Heidelberg (2005)
8. Chepoi, V.D., Dragan, F.F., Vaxes, Y.: Distance and routing labeling schemes for non-positively curved plane graphs. Journal of Algorithms 61(2), 60–88 (2006)
9. Chung, F.R.K.: Universal graphs and induced-universal graphs. Journal of Graph Theory 14, 443–454 (1990)
10. Dragan, F.F., Lomonosov, I.: On compact and efficient routing in certain graph classes. In: Fleischer, R., Trippen, G. (eds.) ISAAC 2004. LNCS, vol. 3341, pp. 402–414. Springer, Heidelberg (2004)
11. Fraigniaud, P., Gavoille, C.: Routing in trees. In: Orejas, F., Spirakis, P.G., van Leeuwen, J. (eds.) ICALP 2001. LNCS, vol. 2076, pp. 757–772. Springer, Heidelberg (2001)
12. Gavoille, C., Ly, O.: Distance labeling in hyperbolic graphs. In: Deng, X., Du, D.-Z. (eds.) ISAAC 2005. LNCS, vol. 3827, pp. 1071–1079. Springer, Heidelberg (2005)
13. Gavoille, C., Paul, C.: Optimal distance labeling schemes for interval and circular-arc graphs. In: Di Battista, G., Zwick, U. (eds.) ESA 2003. LNCS, vol. 2832, pp. 254–265. Springer, Heidelberg (2003)
14. Gavoille, C., Peleg, D.: Compact and localized distributed data structures. Distributed Computing 16, 111–120 (2003) PODC 20-Year Special Issue
15. Gavoille, C., Peleg, D., Pérennès, S., Raz, R.: Distance labeling in graphs. Journal of Algorithms 53, 85–112 (2004)
16. Gilbert, J.R., Rose, D.J., Edenbrandt, A.: A separator theorem for chordal graphs. SIAM Journal on Algebraic and Discrete Methods 5, 306–313 (1984)
17. Kannan, S., Naor, M., Rudich, S.: Implicit representation of graphs. In: 20^{th} Annual ACM Symposium on Theory of Computing (STOC), pp. 334–343. ACM Press, New York (1988)
18. Korman, A., Kutten, S., Peleg, D.: Proof labeling system. In: 24^{th} Annual ACM Symposium on Principles of Distributed Computing (PODC), pp. 9–18. ACM Press, New York (2005)
19. Korman, A., Peleg, D.: Compact separator decompositions in dynamic trees and applications to labeling schemes. In: 21^{st} International Symposium on Distributed Computing (DISC). LNCS, vol. 4731, Springer, Heidelberg (2007)
20. Korman, A., Peleg, D., Rodeh, Y.: Constructing labeling schemes through universal matrices. In: Asano, T. (ed.) ISAAC 2006. LNCS, vol. 4288, pp. 409–418. Springer, Heidelberg (2006)
21. Thorup, M., Zwick, U.: Compact routing schemes. In: 13^{th} Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA), pp. 1–10. ACM Press, New York (2001)

Fast Evaluation of Union-Intersection Expressions

Philip Bille, Anna Pagh, and Rasmus Pagh

Computational Logic and Algorithms Group,
IT University of Copenhagen, Denmark
{beetle,annao,pagh}@itu.dk

Abstract. We show how to represent sets in a linear space data structure such that expressions involving unions and intersections of sets can be computed in a worst-case efficient way. This problem has applications in e.g. information retrieval and database systems. We mainly consider the RAM model of computation, and sets of machine words, but also state our results in the I/O model. On a RAM with word size w, a special case of our result is that the intersection of m (preprocessed) sets, containing n elements in total, can be computed in expected time $O(n(\log w)^2/w + km)$, where k is the number of elements in the intersection. If the first of the two terms dominates, this is a factor $w^{1-o(1)}$ faster than the standard solution of merging sorted lists. We show a cell probe lower bound of time $\Omega(n/(wm \log m) + (1 - \frac{\log k}{w})k)$, meaning that our upper bound is nearly optimal for small m. Our algorithm uses a novel combination of approximate set representations and word-level parallelism.

1 Introduction

Algorithms and data structures for sets play an important role in computer science. For example, the relational data model, which has been the dominant database paradigm for decades, is based on set representation and manipulation. Set operations also arise naturally in connection with database queries that can be expressed as a boolean combination of simpler queries. For example, search engines report documents that are present in the intersection of several sets of documents, each corresponding to a word in the query. If we fix the set of documents to be searched, it is possible to spend time on preprocessing all sets, to decrease the time for answering queries.

The search engine application has been the main motivation in several recent works on computing set intersections [14, 5, 13]. All these papers assume that elements are taken from an ordered set, and are accessed through comparisons. In particular, creating the canonical representation, a sorted list, is the best possible preprocessing in this context. The comparison-based model rules out some algorithms that are very efficient, both in theory and practice. For example, if the preprocessing produces a hashing-based dictionary for each set, the intersection of two sets S_1 and S_2 can be computed in expected time $O(\min(|S_1|, |S_2|))$. This

T. Tokuyama (Ed.): ISAAC 2007, LNCS 4835, pp. 739–750, 2007.

is a factor $\Theta(\log(1 + \max(\frac{|S_1|}{|S_2|}, \frac{|S_2|}{|S_1|})))$ faster than the best possible worst-case performance of comparison-based algorithms.

In this paper we investigate non-comparison-based techniques for evaluating expressions involving unions and intersections of sets on a RAM. (In the search engine application this corresponds to expressions using AND and OR operators.) Specifically, we consider the situation in which each set is required to be represented in a linear space data structure, and propose the *multi-resolution set representation*, which is suitable for efficient set operations. We show that it is possible in many cases to achieve running time that is sub-linear in the total size of the input sets and intermediate results of the expression. For example, we can compute the intersection of a number of sets in a time bound that is sub-linear in the total size of the sets, plus time proportional to the total number of input elements in the intersection. In contrast, all previous algorithms that we are aware of take at least linear time in the worst case over all possible input sets, even if the output is the empty set. The time complexity of our algorithm improves as the word size w of the RAM grows. While the typical word size of a modern CPU is 64 bits, modern CPU design is *superscalar* meaning that several independent instructions can be executed in parallel. This means that in most cases (with the notable exception of multiplication) it is possible to *simulate* operations on larger word sizes with the same (or nearly the same) speed as operations on single words. We expect that word-level parallelism may gain in importance, as a way of making use of the increasing parallelism of modern processor architectures.

1.1 Related Work

Set Union and Intersection. The problem of computing intersections and unions (as well as differences) of sorted sets was recently considered in a number of papers (e.g. [14, 5]) in an *adaptive* setting. A good adaptive algorithm uses a number of comparisons that is close (or as close as possible) to the size of the smallest set of comparisons that determine the result. In the case of two sorted sets, this is the number of interleavings when merging the sets. In the worst case this number is linear in the size of the sets, in which case the adaptive algorithm performs no better than standard merging. However, adaptive algorithms are able to exploit "easy" cases to achieve smaller running time. Mirzazadeh in his thesis [17] extended this line of work to arbitrary expressions with unions and intersections. These results are incomparable to those obtained in this paper: Our algorithm is faster for most problem instances, but the adaptive algorithms are faster in certain cases. It is instructive to consider the case of computing the intersection of two sets of size n where the size of the intersection is relatively small. In this case, an optimal adaptive algorithm is faster than our algorithm only if the number of *interleavings* of the sorted lists (i.e., the number of sublists needed to form the sorted list of the union of the sets) is less than around n/w.

Another idea that has been studied is, roughly speaking, to exploit asymmetry. Hwang and Lin [15] show that merging two sorted lists S_1 and S_2 requires $\Theta(|S_1| \log(1 + \frac{|S_2|}{|S_1|}))$ comparisons, for $|S_1| < |S_2|$, in the worst case over all

input lists. This is significantly less than $O(|S_1| + |S_2|)$ if $|S_1| \ll |S_2|$. This result was generalized to computation of general expressions involving unions and intersections of sets by Chiniforooshan et al. [13]. Given an expression, and the sizes of the input sets, their algorithm uses a number of comparisons that is asymptotically equal to the minimum number of comparisons required in the *worst case* over all such sets.[1] The bounds stated in [13] do not involve the size of the output, meaning that they pessimistically assume the output to be the largest possible, given the expression and the set sizes. In contrast, our bounds will be *output sensitive*, i.e., involve also the size of the result of the expression. We further compare our result to that of [13] in section 1.2.

Approximate Set Representations. There has been extensive previous work on approximate set representations, mainly motivated by applications in networking and distributed systems [8]. Much of this work builds upon the seminal paper on Bloom filters [7]. A Bloom filter for a set S is an approximate representation of S in the sense that for any $x \notin S$ the filter can be used to determine that $x \notin S$ with probability close to 1. However, for an ϵ fraction of elements not in S, called *false positives*, the Bloom filter is consistent with a set that includes these elements. The advantage of allowing some false positives, rather than storing S exactly, is that the space usage drops to around $O(n \log(1/\epsilon))$ bits, practically independent of the size of the universe of which S is a subset. Two Bloom filters for sets S_1 and S_2 can be combined to form a Bloom filter for $S_1 \cap S_2$ (resp. $S_1 \cup S_2$), in a very simple way: By taking bitwise AND (resp. OR) of the data structures.

Bloom filters have been used in connection with computation of relational joins, which are essentially multiset intersections, in the I/O model of computation. The idea is to use a Bloom filter for the smaller set to efficiently find most elements of the larger set that are *not* in the intersection. If the Bloom filter can fit into internal memory, this is a highly efficient procedure for reducing the amount of data that needs to be considered in the join. The algorithm presented in this paper also uses approximate set representations to eliminate elements that will not contribute to the result. However, using Bloom filters does not appear to yield an efficient solution, essentially because the information pertaining to a particular element of S is distributed across the data structure. This makes it hard to locate the set of input elements represented by a particular Bloom filter. Instead, we use the approximate set representation of Carter et al. [11] (see also [18]), which consists of storing, in a compact way, the image of the set under a universal hash function.

1.2 Setup and Results

We consider fully parenthesized expressions with binary operators. That is, we have a rooted binary tree with input sets at the leaves and internal nodes

[1] After personal communication with the authors, we have had confirmed that the algorithm described in [13] is not optimal in certain cases. Specifically, it does not always compute the union of sets in the optimal bound. However, the authors have informed us that the algorithm can be slightly modified to remove this problem.

corresponding to union and intersection operations. Given the sizes of all input sets, we may associate with any node v two numbers (notation from [13]):

- $\psi(v)$ is the maximum possible number of elements in the subexpression rooted at v. (Can be computed bottom-up by summing child values at union nodes, and choosing the minimum child value at intersection nodes.)
- $\psi^*(v)$ is the maximum possible number of elements in the subexpression rooted at v that can appear in the final result. This is the minimum value of $\psi(v)$ on the path from v to the root.

We denote by V the set of nodes in the expression (internal as well as leaves), and let v_0 denote the root node.

Theorem 1. *Given suitably preprocessed sets of total size n, we can compute the value of an expression with binary union and intersection operators in expected time $O(k' + \sum_{v \in V} \lceil \frac{\psi^*(v)}{w} \log^2(\frac{nw}{\psi^*(v)}) \rceil)$, where k' is the number of occurrences in the input of elements in the result. Preprocessing of a set of size n_1 uses linear space and expected time $O(n_1 \log w)$.*

Theorem 1 requires some effort to interpret. We will first state some special cases of the result, and then discuss the general result towards the end of the section. It is not hard to see that the terms in the sum of Theorem 1 corresponding to intersection nodes do not affect the asymptotic value. That is, we could alternatively sum over the set of leaf nodes and union nodes in the expression. In the case where the expression is an intersection of m sets we can further improve our algorithm and analysis to get the following result:

Theorem 2. *Given m preprocessed sets of total size n, we can compute the intersection of the sets in expected time $O(n \log^2 w/w + km)$, where k is the number of elements in the intersection.*

We show the following lower bound, implying that the time complexity of Theorem 2 is within a factor $(\log w)^2 m \log m$ of optimal, assuming $w = (1+\Omega(1)) \log n$. Our lower bound applies to the class of functions whose union-intersection expression has an intersection operation on any root-to-leaf path (an element needs to be in at least two input sets to appear in the result). Note that if there is a path consisting of only union operations, there exists a set where all elements must be included in the result, so this requirement is no serious restriction.

Theorem 3. *Let f be a function of m sets given by a union-intersection expression with an intersection node on any root-to-leaf path. For integers n and $k \leq n/m$, any (randomized) algorithm in the cell probe model that takes representations of sets $S_1, \ldots, S_m \subseteq \{0,1\}^w$, where $\sum_i |S_i| \leq n$ and $|f(S_1, \ldots, S_m)| \leq k$, and computes $|f(S_1, \ldots, S_m)|$ must use expected time at least $\Omega(n/(wm \log m) + (1 - \frac{\log k}{w})k)$ on a worst-case input. The lower bound holds regardless of how the sets are represented.*

Possibly the best way of understanding the general result in Theorem 1 is to compare the complexity to the comparison-based algorithm of [13]. Though it

might not result in the best running time for our algorithm, we make the comparison in the case where any group of adjacent union operators is arranged as a perfectly balanced tree in the expression tree (we could modify our algorithm to always make this change to the expression). The algorithm of [13] takes an expression where operators have unbounded degree, and where union and intersection nodes alternate. It can be applied in our setting by combining groups of adjacent union and intersection operators. The time usage is *at least* $\Omega(k' + \sum_{v \in V} \psi^*(v))$ (in fact, the complexity also involves a logarithmic factor on each term, but it is not easily comparable to the factor in our result). Thus, if the word length is sufficiently large, e.g. $w = (\log n)^{\omega(1)}$, our algorithm gains a factor $w^{1-o(1)}$ compared to [13].

We observe that all our results immediately imply nontrivial results in the I/O model [1]. For the upper bounds, this is because any RAM algorithm can be simulated in the same I/O bound as long as w is bounded by the number of bits in a disk block. In other words, if B is the number of words in a disk block, we can get I/O bounds by replacing w by Bw in the results. In fact, the power of 2 in the bounds can be reduced to 1 in this setting, as the I/O model does not count the cost of computation. Our lower bound also holds in the I/O model, with w replaced by Bw, independently of the size of internal memory. (The same proof applies.)

1.3 Technical Overview

Our results are obtained through non-trivial combination of several known techniques. We use the idea of Carter el al. [11] to obtain an approximate representation of a set by storing a set $h(S)$ of hash function values rather than the set S itself. Storing the approximation in a naïve way (using at least $\log n$ bits per element) does not lead to a significant speedup in general. Instead, a compact representation of the set $h(S)$ is needed. We use a bucketed set representation, as in the dictionary of Brodnik and Munro [9], to get a compact representation of $h(S)$ that is suitable for word-parallel set operations. Specifically, we show how set operations on small integers packed in words can be efficiently implemented, using ideas from [2, 3]. This allows us to quickly approximate the intersection of any two sets in the sense that we get a compressed list of references to the elements in the intersection plus a small fraction of the elements not in the intersection. To compute the intersection we compute the intersection of the subsets of "candidates" in the standard way, using hashing. The generalization to the case of expressions involving arbitrary unions and intersections is an extension of this idea, using a variant of a technique from [13] to keep the sizes of the sets we have to deal with as small as possible. Our lower bound is shown by a reduction to multi-party communication complexity.

2 Main Algorithm and Data Structure

In this section we present most of our algorithm and data structure, postponing the material on word-level parallelism to Section 3 (which is used as a blackbox

in this section). Specifically, we show how to reduce the problem of performing unions and intersections on sets of words to the problem of performing these operations on sets from a smaller universe. Due to space constraints we refer to the technical report version of this paper [6] for the time and space analysis.

2.1 Overview of Special Case: Intersection

We first present the main ideas in the case where the expression is an intersection of m sets. The basis of the approach is to map elements of $\{0,1\}^w$ to a smaller universe using a hash function h, and compute the intersection $H = h(S_1) \cap \cdots \cap h(S_m)$. Now, if $x \in S_1 \cap \cdots \cap S_m$ then $h(x) \in H$. On the other hand, if $x \notin S_1 \cap \cdots \cap S_m$ then, if h is suitably chosen, we will have $h(x) \notin H$ with probability close to 1. Thus, we can regard H as representing a good approximation of $S_1 \cap \cdots \cap S_m$. In particular, if we compute the sets $S_i' = \{x \in S_i \mid h(x) \in H\}$, $i = 1, \ldots, m$, we expect that S_i' does not contain many elements of $S_i \setminus (S_1 \cap \cdots \cap S_m)$. Since $S_i \supseteq S_i' \supseteq S_1 \cap \cdots \cap S_m$ we can compute the intersection of $S_1, \ldots, S_m$ as $S_1' \cap \cdots \cap S_m'$ — using a standard linear time hashing-based algorithm. The challenge of this approach is to keep the cost of computing H and the sets S_i' low. We store preprocessed, compressed representations of the sets $h(S_i)$ using only $O(\log w)$ bits per hash value, which allows us to compute H in time that is sub-linear in the size of the input sets. The elements of S_i' are extracted in additional time $O(|S_i|)$. The details of these steps appear in sections 2.3 and 3. Readers mainly interested in the case of computing a single intersection may skip the description of the general case in the next subsection.

2.2 The General Case

In the rest of the paper we let f denote the function of m input sets given by the expression to be evaluated. Since $f(S_1, \ldots, S_m)$ is monotone in the sense that adding an element to an input set can never remove an element from $f(S_1, \ldots, S_m)$ we have that for any $x \in f(S_1, \ldots, S_m)$ it holds that $h(x) \in f(h(S_1), \ldots, h(S_m))$. This means we can compute $f(S_1, \ldots, S_m)$ by the following steps:

1. Compute $H = f(h(S_1), \ldots, h(S_m))$.
2. For all i compute the set $S_i' = \{x \in S_i \mid h(x) \in H\}$.
3. Compute $f(S_1', \ldots, S_m')$ to get the result.

We will show how, starting with a suitable, compressed representation of the sets $h(S_1), \ldots, h(S_m)$, we can efficiently perform the first two steps such that the sets S_i' are significantly smaller than the S_i in the following sense: *Most* of the elements that do not occur in $f(S_1, \ldots, S_m)$ have been removed. This means that, except for negligible terms, the time for performing the third step, using the standard linear time hashing-based algorithm, depends on the number of input elements in the output rather than on the size of the input. Conceptually, the first step computes the expression on approximate representations of the

sets $S_1, \ldots, S_m$. Then the information extracted from this is used to create a smaller problem instance with the same result, which is then used to produce the answer.

Assume for now that h is given, and that we have access to data structures for $h(S_1), \ldots, h(S_m)$. The details on how to choose h appear in Section 2.3. The computation of $f(h(S_1), \ldots, h(S_m))$ is done bottom-up in the expression tree in the same order as the algorithm of [13]: For an intersection node v we first recursively process the child subtree whose root has the smallest value of ψ^* — the children of union nodes are processed recursively in arbitrary order. We adopt another idea of [13]: If the set computed for the subtree rooted at v has size more than $2\psi^*(v)$, we reduce the size of the set to at most $\psi^*(v)$ by computing the intersection with the smallest child set of an intersection node on the path from v to the root. Observe that this will only remove elements that are not in the output. Due to the way we traverse the expression tree, the relevant child set will already have been computed. For every node v in the expression tree, we store the result $\mathcal{I}_v$ of the subexpression rooted at v.

For the root node v_0 define $\mathcal{I}'_{v_0} = \mathcal{I}_{v_0}$. To compute the sets S'_i we first traverse the tree top-down and compute for every non-root node v the intersection $\mathcal{I}'_v = \mathcal{I}'_v \cap \mathcal{I}'_{p(v)}$, where $p(v)$ is the parent node of v. Observe that, by induction, $\mathcal{I}'_v = \mathcal{I}_v \cap f(h(S_1), \ldots, h(S_m))$. We will see that the time for this procedure is dominated by the time for computing $f(h(S_1), \ldots, h(S_m))$. At the end we have computed $h(S'_i) = f(h(S_1), \ldots, h(S_m)) \cap h(S_i)$ for all i. All that remains is to find the corresponding elements of S'_i, which is easily done by looking up the hash function values in a hash table that stores $h(S_i)$ with the corresponding elements of S_i as satellite information.

Finally, we compute $f(S'_1, \ldots, S'_m)$ by first identifying all duplicate elements in the sets (by inserting them in a common hash table), keeping track of which set each element comes from. Then for each element decide whether it is in the output by evaluating the expression. This can be done in time proportional to the number of occurrences of the element: First annotate each leaf and intersection node in the expression tree with the nearest ancestor that is an intersection node. Then compute the set corresponding to each intersection node bottom-up. The time spent on an intersection node is bounded by the total size of the sets at intersection nodes immediately below it, but the intersection of these sets has size at most half of the total size. This implies the claimed time bound by a simple accounting argument.

2.3 Data Structure

The best choice of h depends on the particular expression and size of input sets. For example, when computing the intersection $S_1 \cap S_2$ we want the range of h to have size significantly larger than the smaller set (S_1, say). This will imply that most elements in $h(S_2 \backslash S_1)$ will not be in $h(S_1)$, and there will be a significant reduction of the problem instance in step 2 of the main algorithm. On the other hand, the time and space usage grows with the size of the range of the hash function used, so it should be chosen no larger than necessary. In conclusion, to

be able to choose the most suitable one in a given situation, we wish to store the image of every set under several hash functions, differing in the size of their range. The images of the set under various hash functions can be thought of as representations of the set at different resolutions. Hence, we name our data structure the *multi-resolution set representation*. As we show in the full version of this paper [6], it suffices to use a hash function with range $\{0,1\}^r$, where $r = \log n + O(\log w)$ and n is the total size of the input sets.

The hash functions will all be derived from a single "mother" hash function h^*, a strongly universal hash function [10, 19] with values in the range $\{0,1\}^w$. This is a global hash function that is shared for all sets. The hash function h_r, for $1 \leq r \leq w$ is defined by $h_r(x) = h^*(x) \operatorname{div} 2^{w-r}$, where "div" denotes integer division (we use the natural correspondence between bit strings and nonnegative integers). Note that h_r has function values of r bits. To store $h_r(S)$ for a particular set S, $r \geq \log|S|+1$, requires $|h_r(S)|(r-\log|h_r(S)|+\Theta(1))$ bits, by information theoretical arguments. Since we may have $|h_r(S)| = |S|$ the space usage could be as high as $|S|(r-\log|S|+\Theta(1))$. Note that the required space per element is constant when $r \leq \log|S| + O(1)$, and then grows linearly with r.

If we store $h_r(S)$ for all r, $\log|S| < r \leq w$, the space usage may be $\Omega(w)$ times that of storing S itself. To achieve linear space usage we store $h_r(S)$ only for selected values of r, depending on $|S|$, namely $r \in \{\lceil \log|S| \rceil + 2^i \mid i = 0,1,2,\ldots,\lfloor \log(w-\log|S|) \rfloor\}$. These sets are stored using the bucketed set representation of Section 3 which gives a space usage for $h_r(S)$ of $O(|S|(r-\log|S|+\log w))$ bits. To get the representation of $h_r(S)$ for arbitrary r we access the stored representation of $h_{r'}$, where $r' > r$, and throw away the $r'-r$ least significant bits of its elements (see Section 3 for details). Choosing r' as small as possible minimizes the time for this step. We build the bucketed set representation of the largest value of r in $O(|S|)$ time by hashing, and then apply Lemma 4 iteratively to get the structures for the lower values of r.

The final thing we need is a hash table that allows us to look up a value $h_r(x)$ and retrieve the element(s) in S that have this value of h_r. This can be done by using the $\lceil \log|S| \rceil$ most significant bits of h_r as index into a chained hash table. Since the values of these bits are common for all h_r, $\log|S| < r \leq w$, we only need to store a single hash table. Note that the size of the hash table is $\Theta(|S|)$, which means that the expected lookup time is constant.

3 Bucketed and Packed Sets

We describe two representations of sets of elements from a small universe and provide efficient algorithms for computing union and intersection in the representations. Proofs of the lemmas in this section can be found in the full version [6].

3.1 Packed Sets

Given a parameter f we partition words into $k = w/(f+1)$ substrings, called *fields*, numbered from right to left. The most significant bit of a field is called

the *test bit* and the remaining f-bits are called the *entry*. A word is viewed as an array A capable of holding up to k bit strings of length f. If the ith test bit is 1 we consider the ith field to be *vacant*. Otherwise the field is *occupied* and the bit string in the ith entry is interpreted as the binary encoding of a non-negative integer. If $|A| > k$ we can represent it in $\lceil |A|/k \rceil$ words; each storing up to k elements. We call an array represented in this way a *packed array with parameter* f (or simply *packed array* if f is understood from the context). For our purposes we will always assume that fields are capable of storing the total number of fields in a word, that is, $f \geq \log k$. In the following we present a number of useful ways to manipulate packed arrays.

Suppose A is a packed array containing x occupied fields. Then, *compacting* A means moving all the occupied fields into the first x fields of A while maintaining the order among them.

Lemma 1 (Andersson et al. [3]). *A packed array A with parameter f can be compacted in $O\left(|A| \left\lceil f^2/w \right\rceil\right)$ time.*

Let $X = x_1, \ldots, x_m$ be a sequence of f-bit integers. If X is given as a packed array with parameter f, such that the ith field, $1 \leq i \leq m$, holds x_i, we say that X is a *packed sequence with parameter* f. We use the following result:

Lemma 2 (Albers and Hagerup [2]). *Two sorted packed sequences X_1 and X_2 with parameter f can be merged into a single sorted packed sequence in $O\left((|X_1| + |X_2|) \left\lceil f^2/w \right\rceil\right)$ time.*

We refer to a sorted, packed sequence of integers as a *packed set*.

Lemma 3. *Given packed sets S_1 and S_2 with parameter f, the packed sets $S_1 \cup S_2$ and $S_1 \cap S_2$ with parameter f can be computed in $O\left((|S_1| + |S_2|) \left\lceil f^2/w \right\rceil\right)$ time.*

3.2 Bucketed Sets

Let S be a set of l-bit integers. For a given parameter $b \leq l$ we partition S into 2^b subsets, $S_0, \ldots, S_{2^b-1}$, called *buckets*. Bucket S_i contain all values in the range $[2^{i(l-b)}, 2^{(i+1)(l-b)} - 1]$, and therefore all values in S_i agree on the b most significant bits. Hence, to represent S_i it suffices to know the b most significant bits together with the set of the $l - b$ least significants bits. We can therefore compactly represent S by an array of length 2^b, where the ith entry points to the packed set (with parameter $l - b$) of the $l - b$ least significant bits of S_i. We say that S is a *bucketed set with parameter* b if it is given in this representation. Note that such an encoding of S uses $O(2^b w + |S|(l - b))$ bits. As above, we assume that fields in packed sets are capable of holding the number of fields in a word, that is, we assume that $(l - b) \geq \log w - \log(l - b + 1))$ in any bucketed set. We need the following results to manipulate bucketed sets.

Lemma 4. *Let S be a bucketed set of l-bit integers with parameter b. Then,*

1. *Given an integer b' we can convert S into a bucketed set with parameter b' in time $O\left(2^{\max(b,b')} + |S| \left\lceil (l - \min(b, b'))^2/w \right\rceil\right)$.*

2. *Given an integer $b < x \leq l$ we can compute the bucketed set $S' = \{j \operatorname{div} 2^x \mid j \in S\}$ of $l - x$ bit integers with parameter b in $O\left(2^b + |S| \left\lceil (l-b)^2/w \right\rceil\right)$ time.*

Let S be a bucketed set of l-bit integers with parameter b. We say that S is a *balanced bucketed set* if b is the largest integer such that $b \leq \log |S| - \log w$. Intuitively, this choice of b balances the space for the array of buckets and the packed sets representing the buckets. Since $l \geq \log |S|$ the condition implies that $l - b \geq l - \log |S| + \log w \geq \log w - \log(l - b + 1)$. Hence, the field length of the packed sets representing the buckets in S is as required. Also, note that the space for a balanced bucketed set S is $O(2^b w + |S|(l-b)) = O(|S|(l - \log |S| + \log w))$.

Lemma 5. *Let S_1 and S_2 be balanced bucketed sets of l-bit integers. The balanced bucketed sets $S_1 \cup S_2$ and $S_1 \cap S_2$ can be computed in time*

$$O\left((|S_1| + |S_2|) \left\lceil (l - \log(|S_1| + |S_2|) + \log w)^2/w \right\rceil\right) \quad .$$

If $l = \Theta(\log(|S_1| + |S_2|))$ Lemma 5 provides a speedup by a factor of $w/\log^2 w$.

4 Lower Bound

In this section we show Theorem 3. The proof uses known bounds from *t-party communication complexity*, where t communicating players are required to compute a function of n-bit strings $x_1, \ldots, x_t$, where x_i is held by player i, using as little communication as possible. We consider the *blackboard model* where a bit communicated by one player is seen by all other players, and consider the following binary functions:

EQ(x_1, x_2) which has value 1 iff $x_1 = x_2$. (Here $t = 2$.)

DISJ$_{n,t}(x_1, \ldots, x_t)$ which has value 1 iff there is no position where two bit strings x_i and x_j both have a 1 (i.e., all pairs are "disjoint"). We consider this problem under the *unique intersection assumption*, where either all pairs are disjoint, or there exists a single position where all bit strings have a 1. We allow the protocol to behave in any way if this is not the case.

Solving **EQ** exactly requires communication of $\Omega(n)$ bits, for both deterministic and randomized protocols [20, 16]. That is, the trivial protocol where one player communicates her entire bit string is optimal. Chakrabarti at al. [12], based on work by Bar-Yossef et al. [4], showed that solving **DISJ**$_{n,t}$ exactly requires $\Omega(n/(t \log t))$ bits of communication in expectation, even under the unique intersection assumption and when the protocol is randomized.

Our main observation is that if sets $S_1, \ldots, S_t$ have been independently preprocessed, we can view any algorithm that computes $f(S_1, \ldots, S_t)$ as a communication protocol where each player holds a set. Whenever the algorithm accesses the representation of S_i it corresponds to w bits being sent by player i. Formally, given any (possibly randomized) algorithm that computes $|f(S_1, \ldots, S_t)|$, where $S_1, \ldots, S_t$ have been individually preprocessed in an arbitrary way, we derive

communication protocols for **EQ** and $\mathbf{DISJ}_{n,t}$, and use the lower bounds for these problems to conclude a lower bound on the expected number of steps used by the algorithm. We note that this reduction from communication complexity is different from the reduction from asymmetric communication complexity commonly used to show data structures lower bounds.

Let n and k, $1 \leq k \leq n/t$, denote integers such that the algorithm correctly computes $|f(S_1, \ldots, S_t)|$ provided that the sum of sizes of the sets is at most $n+1$, and that $|f(S_1, \ldots, S_t)| \leq k$. Let τ denote the number of cell probes on a worst-case input of this form. Given vectors $x_1, \ldots, x_t \in \{0,1\}^n$ satisfying the unique intersection assumption, we consider the sets $S_i = \{j \mid x_i \text{ has a 1 in position } j\}$ and their associated representations (which could be chosen in a randomized fashion). Observe that the total size of the sets is at most $n+1$, and that $|f(S_1, \ldots, S_t)| = 0$ if and only $S_1, \ldots, S_t$ are disjoint (using the assumptions on f). By simulating the algorithm on these representations, we get a communication protocol for **DISJ** using τw bits in expectation. By the lower bound on $\mathbf{DISJ}_{n,t}$ we thus have $\tau w = \Omega(n/(t \log t))$ on a worst case input, i.e., $\tau = \Omega(n/(wt \log t))$ cell probes are needed.

Consider the function $f'(S_1, S_2) = f(S_1, \ldots, S_1, S_2)$. Clearly, a lower bound on the cost of computing f' applies to f as well. We denote by $\binom{\{0,1\}^w}{k}$ the set of subsets of $\{0,1\}^w$ having size k. Let $q = \lfloor \log_2 |\binom{\{0,1\}^w}{k}| \rfloor$, and let ϕ be any injective function from $\{0,1\}^q$ to $\binom{\{0,1\}^w}{k}$. Given two vectors $x, y \in \{0,1\}^q$ we consider the sets $S_1 = \phi(x)$ and $S_2 = \phi(y)$, which satisfy $|f'(S_1, S_2)| \leq k$ and $(t-1)|S_1| + |S_2| \leq n$. Since ϕ is injective, we have that $x = y$ iff $|f'(S_1, S_2)| = k$. Thus, similar to above we get a communication protocol for **EQ** that uses τw bits in expectation on a worst-case input. By the lower bound on **EQ** we have $\tau = \Omega(q/w)$, implying that $\tau = \Omega(k(w - \log_2 k)/w)$. The maximum of our two lower bounds is a factor of at most two from the sum stated in the theorem, finishing the proof.

5 Conclusion and Open Problems

We have shown how to use two algorithmic techniques, approximate set representations and word-level parallelism, to accelerate algorithms for basic set operations. Potentially, the results (or techniques) could have a number of applications in problem domains such as databases (relational, textual,...) where some preprocessing time (indexing) may be invested to keep the cost of queries low.

It is an interesting problem whether our results can be extended to handle non-monotone set operators such as set difference. The technical problem here is that one would have to deal with two-sided errors in the estimates of the intermediate results.

Acknowledgement. We thank Mikkel Thorup for providing us useful insight on the use of word-level parallelism on modern processors.

References

1. Aggarwal, A., Vitter, J.S.: The input/output complexity of sorting and related problems. Comm. ACM 31(9), 1116–1127 (1988)
2. Albers, S., Hagerup, T.: Improved parallel integer sorting without concurrent writing. Information and Computation 136, 25–51 (1997)
3. Andersson, A., Hagerup, T., Nilsson, S., Raman, R.: Sorting in linear time? In: Proceedings of the 27th annual ACM symposium on Theory of computing (STOC 1995), pp. 427–436 (1995)
4. Bar-Yossef, Z., Jayram, T.S., Kumar, R., Sivakumar, D.: An information statistics approach to data stream and communication complexity. J. Comput. System Sci. 68(4), 702–732 (2004)
5. Barbay, J., Kenyon, C.: Adaptive intersection and t-threshold problems. In: Proceedings of the 13th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2002), pp. 390–399 (2002)
6. Bille, P., Pagh, A., Pagh, R.: Fast evaluation of union-intersection expressions. Technical Report arXiv:0708.3259v1 (2007), `http://arxiv.org`
7. Bloom, B.H.: Space/time trade-offs in hash coding with allowable errors. Communications of the ACM 13(7), 422–426 (1970)
8. Broder, A.Z., Mitzenmacher, M.: Network applications of Bloom filters: A survey. In: Proceedings of the 40th Annual Allerton Conference on Communication, Control, and Computing, pp. 636–646. ACM Press, New York (2002)
9. Brodnik, A., Munro, J.I.: Membership in constant time and almost-minimum space. SIAM J. Comput. 28(5), 1627–1640 (1999)
10. Carter, J.L., Wegman, M.N.: Universal classes of hash functions. J. Comput. System Sci. 18(2), 143–154 (1979)
11. Carter, L., Floyd, R., Gill, J., Markowsky, G., Wegman, M.: Exact and approximate membership testers. In: Proceedings of the 10th Annual ACM Symposium on Theory of Computing (STOC 1978), pp. 59–65. ACM Press, New York (1978)
12. Chakrabarti, A., Khot, S., Sun, X.: Near-optimal lower bounds on the multi-party communication complexity of set disjointness. In: IEEE Conference on Computational Complexity, pp. 107–117. IEEE Computer Society Press, Los Alamitos (2003)
13. Chiniforooshan, E., Farzan, A., Mirzazadeh, M.: Worst case optimal union-intersection expression evaluation. In: Caires, L., Italiano, G.F., Monteiro, L., Palamidessi, C., Yung, M. (eds.) ICALP 2005. LNCS, vol. 3580, pp. 179–190. Springer, Heidelberg (2005)
14. Demaine, E.D., López-Ortiz, A., Munro, J.I.: Adaptive set intersections, unions, and differences. In: Proceedings of the 11th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2000), pp. 743–752 (2000)
15. Hwang, F.K., Lin, S.: A simple algorithm for merging two disjoint linearly ordered sets. SIAM J. Comput. 1(1), 31–39 (1972)
16. Kushilevitz, E., Nisan, N.: Communication Complexity. Cambridge University Press, Cambridge (1997)
17. Mirzazadeh, M.: Adaptive comparison-based algorithms for evaluating set queries. Master's thesis, University of Waterloo (2004)
18. Pagh, A., Pagh, R., Rao, S.S.: An optimal Bloom filter replacement. In: Proceedings of the 16th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2005), pp. 823–829. ACM Press, New York (2005)
19. Thorup, M.: Even strongly universal hashing is pretty fast. In: Proceedings of the 11th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2000), pp. 496–497. ACM Press, New York (2000)
20. Yao, A.C.: Some complexity questions related to distributive computing (preliminary report). In: Proceedings of the 11th Annual ACM Symposium on Theory of Computing (STOC 1979), pp. 209–213. ACM Press, New York (1979)

A Sub-cubic Time Algorithm for the *k*-Maximum Subarray Problem

Sung Eun Bae and Tadao Takaoka

Department of Computer Science, University of Canterbury
Christchurch, New Zealand
{seb43,tad}@cosc.canterbury.ac.nz

Abstract. We design a faster algorithm for the k-maximum sub-array problem under the conventional RAM model, based on distance matrix multiplication (DMM). Specifically we achieve $O(n^3\sqrt{\log\log n/\log n} + k\log n)$ for a general problem where overlapping is allowed for solution arrays. This complexity is sub-cubic when $k = o(n^3/\log n)$. The best known complexities of this problem are $O(n^3 + k\log n)$, which is cubic when $k = O(n^3/\log n)$, and $O(kn^3\sqrt{\log\log n/\log n})$, which is sub-cubic when $k = o(\sqrt{\log n/\log\log n})$.

1 Introduction

The maximum subarray (MSA) problem is to compute a rectangular portion in a given two-dimensional (n, n)-array that maximizes the sum of array elements in it. If the array elements are all non-negative we have the trivial solution of the whole array. Thus we normally subtract the mean or median value from each array element. This problem has wide applications in graphics and data mining for marketing, as described in [4].

This problem was first introduced by Grenander and brought to computer science by Bentley [7] with an algorithm of $O(n^3)$. Tamaki and Tokuyama [22] obtained a sub-cubic algorithm based on distance matrix multiplication (DMM), by reducing the problem to DMM and showing that the time complexities of the two problems are of the same order. Takaoka [19] simplified the algorithm for implementation.

The k-maximum subarray (k-MSA) problem is to obtain the maximum sub-array, the second maximum subarray, ..., the k-th maximum subarray in sorted order for k up to $O(n^4)$. We can define two such problems. One is the general case where we allow overlapping portions, and the other is for disjoint portions. We consider the general problem in this paper. Let $M(n)$ be the time complexity for DMM for an (n, n)-matrix. We solve the problem in $O(M(n) + k\log n)$ time for the general problem with an (n, n)-array, where $M(n) = O(n^3\sqrt{\log\log n/\log n})$.

Preceding results for the one-dimensional problem are $O(kn)$ by Bae and Takaoka [1], $O(min(n\sqrt{k}, n\log^2 n))$ by Bengtsson and Chen [5], $O(n\log k)$ by Bae and Takaoka [2], $O(n + k\log n)$ by Bae [4], Cheng, et. al. [11], Bengtsson, et. al. [6], $O(n\log n + k)$ expected time by Lin, et. al. [17], and $O(n + k)$ by Brodal,

T. Tokuyama (Ed.): ISAAC 2007, LNCS 4835, pp. 751–762, 2007.

et. al. [8]. Obviously we can solve the two-dimensional problem by applying the one-dimensional algorithm to all $O(n^2)$ strips of the array, resulting in the time complexity multiplied by $O(n^2)$. For the algorithms specially designed for the two-dimensional case, we have $O(kn^3(\log\log n/\log n)^{1/2})$ by [2] and $O(n^3+k)$ by [8]. The last is for k maximum subarrays in unsorted order.

These results are mainly based on extension of optimal algorithms for the one-dimensional problem to the two-dimensional problem. Our results in this paper and [2] show an extension of an optimal algorithm in one dimension to two dimensions does not produce optimal solutions for the two-dimensional problem.

The best known results for the disjoint case are the straightforward $O(kM(n))$, which is sub-cubic for small k such as $k=o(\sqrt{\log n/\log\log n})$, where $M(n)$ is the time for DMM, and $O(n^3+kn^2\log n)$ by Bae and Takaoka [3] for larger k. The problem here is to find the maximum, the second maximum, etc. from the remaining portion.

In the application of graphics, our problem is to find the brightest spot, second brightest spot, ..., k-th brightest spot. In the application of data mining, suppose we have a sales database with records of sales amount of some commodity with numerical attributes such as age, annual income, etc. Then the rectangular portion of age and annual income in some range that maximizes the amount corresponds to obtaining the association rule that maximizes the confidence that if a person is in the range, then he is most likely to buy the commodity. Similarly we can identify the second most promising customer range, etc.

The computational model in this paper is the conventional RAM, where only arithmetic operations, branching operations, and random accessibility with $O(\log n)$ bits are allowed.

The engine for our problem is an efficient algorithm for DMM. Since a sub-cubic algorithm for DMM was achieved by Fredman [14], there have been several improvements [18], [15], [16], [20], [23], [21], [9], [10]. We modify the algorithm in [18] for DMM whose complexity is $O(n^3\sqrt{\log\log n/\log n})$, and extend it to our problem. The recent improvements for DMM after [18] are slightly better, and it may be possible they can be tuned for speed-up of the k-MSA problem.

The main technique in this paper is tournament. Specifically we reorganize the structure of the maximum subarray algorithm based on divide-and-conquer into a tournament structure, which serves as an upper structure. We also reorganize the DMM algorithm into a tournament, which works as a lower structure. Through the combined tournament, the maximum, second maximum, etc. are delivered in $O(\log n)$ time per subarray.

In section 2, basic definitions of tournaments and DMM are given. In section 3, the $X+Y$ problem is defined and a well-known algorithm for it is described for the later development.

In section 4, we give the definition of the maximum subarray problem and a divide-and-conquer algorithm for it. In section 5, we reorganize the algorithm in section 4 into a tournament style, and explain how to combine it with DMM to solve the k-MSA problem. The $X+Y$ algorithm is used as glue in this combination.

The DMM algorithm used is based on two-level divide-and-conquer. In section 6, the upper division is described. In section 7, the lower division is handled through a table look-up. The table in [18] is enhanced to handle several integers in an encoded form, rather than a single integer, at each table entry.

Section 8 concludes the paper, discussing possibilities for further speed-up and extension of similar ideas to the disjoint problem.

This paper achieves a new time complexity through a combination of known methods and tools. Note that we use the same name k in two different meanings; indexing in arrays, and the k for the k-MSA problem.

2 Basic Definitions

An r-ary tournament T is an r-ary tree such that each internal node has r internal nodes and some external nodes as children, or some external nodes only as children. It also has a key, which originates from itself if it is an external node, or is extracted from one of its children if it is an internal node. Each external node has a numerical datum as a key. External nodes can be regarded as participants of the tournament. A parent has the minimum of those keys of its children. We call this a minimum tournament. A maximum tournament is similarly defined. In other words a parent is the winner among its children. The external nodes form the leaves of the tree. We form a complete r-ary tree as far as internal nodes are concerned. Also a node maintains some identity information of the winner that reached this node, such as the original position of the winner, etc. The key and this kind of information eventually propagates to the root, and the winner is selected. The size of the tournament, defined by the number of nodes, is $O(n)$, if there are n external nodes.

If we use a binary tournament for sorting, the identity can be the position of the data item in the original array. We can build up a minimum tournament for n data items in $O(n)$ time. After that, successive k minima can be chosen in $O(k \log n)$ time. This can be done by replacing the key of the winning item at the bottom level, that is, in a leaf, by ∞ and performing matches along the winning path spending $O(\log n)$ time for the second winner, etc. Thus k minima can be chosen in $O(n + k \log n)$ time in sorted order. If $k = n$, this is a sorting process in $O(n \log n)$ time, called the tournament sort. We use a similar technique of tournament in the k-MSA problem.

The distance matrix multiplication is to compute the following distance product $C = AB$ in (1) for two (n, n)-matrices $A = [a_{ij}]$ and $B = [b_{ij}]$ whose elements are real numbers. We can define (1) with "max" also.

$$c_{ij} = min_{k=1}^{n}\{a_{ik} + b_{kj}\}, (i, j = 1, ..., n) \quad (1)$$

The operation in the right-hand side of (1) is called distance matrix multiplication of the *min* version, and A and B are called distance matrices in this context. The index k that gives the minimum in (1) is called the witness for c_{ij}. If we use *max* instead we call it the *max* version.

Suppose we have a three layered acyclic graph for which A is a connection matrix from layer 1 to layer 2, and B is that from layer 2 to layer 3. Each layer has vertices $1, ..., n$, and the distance from i in layer 1 to j in layer 2 is a_{ij}, and that from layer 2 to layer 3 is b_{ij}. Then c_{ij} is the shortest distance from i in layer 1 to j in layer 3.

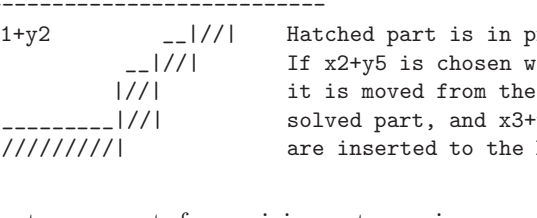
To solve the k-MSA problem, we want to find up to k shortest distances from layer 1 to layer 3 between any vertices. We use this version of extended DMM in this paper, whereas k-DMM in [2] computes k shortest paths for each pair (i, j) with i in layer 1 and j in layer 3, which is rather time consuming. If we solve DMM in $M(n)$ time in such a way that a tournament of some size becomes available for the extended DMM within the same time complexity, then k shortest distances can be found in $O(M(n) + k \log n)$ time for k up to $O(n^3)$, as shown in Sections 6 and 7.

We actually need at most k shortest distances in total for all DMMs used in our k-MSA algorithm, and our requirement is that the newly designed DMM algorithm return the next shortest distance for any pair (i, j), that is, i in layer 1 to j in layer 3, in $O(\log n)$ time.

3 X + Y Problem

Let X and Y be lists of n numbers. We want to choose k smallest numbers from the set $Z = \{x + y | (x \in X) \wedge (y \in Y)\}$. We organize a tournament for each of X and Y in $O(n)$ time. Let the imaginary sorted lists be $X = (x_1, ..., x_n)$ and $Y = (y_1, ..., y_n)$. Actually they are extracted from the tournaments as the computation proceeds. We successively take elements from those sorted lists, one in $O(\log n)$ time. Obviously $x_1 + y_1$ is the smallest. The next smallest is $x_1 + y_2$ or $x_2 + y_1$. Let us have an imaginary two-dimensional array whose (i, j)-element is $x_i + y_j$. As the already selected elements occupy some portion of the top left corner, which we call the solved part, we can prepare a heap to represent the border elements adjacent to the solved region. By keeping selecting minima from the heap, and inserting new bordering elements, we can solve the problem in $O(n + k \log n)$ time.

See the figure below for illustration.



If we change the tournaments from minimum to maximum, we can find k maxima in the same amount of time. Also with similar arrangements, we can select k largest or smallest from $X - Y = \{x - y | (x \in X) \wedge (y \in Y)\}$ in the same amount of time. We use this simple algorithm rather than sophisticated ones such as [13], since these two are equivalent in computing time for k minima in sorted order.

4 The Maximum Subarray Problem

Now we proceed to the maximum subarray problem for an array of size (m, n). The cubic algorithm for this problem given by Bentley [7] was improved to sub-cubic by Tamaki and Tokuyama [22]. We review the simplified sub-cubic version in [19]. We give a two-dimensional array $a[1..m, 1..n]$ of real numbers as input data. The maximum subarray problem is to maximize the sum of the array portion $a[k..i, l..j]$, that is, to obtain the sum and such indices (k, l) and (i, j). We suppose the upper-left corner has co-ordinates (1,1). Bentley's algorithm finds the maximum subarray in $O(m^2n)$ time, which is cubic $O(n^3)$ when $m = n$.

For simplicity, we assume the given array a is a square (n, n)-array. We compute the prefix sums $s[i, j]$ for array portions of $a[1..i, 1..j]$ for all i and j with boundary condition $s[i, 0] = s[0, j] = 0$. Obviously this can be done in $O(n^2)$ time for an (n, n)-array. The outer framework of the algorithm is given below. Note that the prefix sums once computed are used throughout recursion.

Algorithm M: Maximum subarray
1. If the array becomes one element, return its value.
2. Let A_{tl} be the solution for the top left quarter.
3. Let A_{tr} be the solution for the top right quarter.
4. Let A_{bl} be the solution for the bottom left quarter.
5. Let A_{br} be the solution for the bottom right quarter.
6. Let A_{column} be the solution for the column-centered problem.
7. Let $A_{left-row}$ be the solution for the row-centered problem for the left half.
8. Let $A_{right-row}$ be the solution for the row-centered problem for the right half.
9. Let the solution A be the maximum of those seven.

The *coverage* of a solution array is the smallest square region, defined by the above recursive calls, in which the solution is obtained. It is given by index pairs. The *scope* of a solution array is the index pairs $((k, l), (i, j))$ if the solution is $a[k..i, l..j]$. A coverage is also defined by the co-ordinates of the top-left corner, and those of the bottom-right corner. If we call the above algorithm for $a[1..n, 1..n]$, for example, the coverage of A is $((1, 1), (n, n))$, that of A_{tr} is $((1, n/2 + 1), (n/2, n))$, etc.

Here the column-centered problem is to obtain an array portion that crosses over the central vertical line with maximum sum, and can be solved in the following way. $A_{left-row}$ and $A_{right-row}$ can be computed similarly.

$$A_{column} = max_{k=0,l=0,i=1,j=n/2+1}^{i-1,n/2-1,n,n} \{s[i, j] - s[i, l] - s[k, j] + s[k, l]\}.$$

In the above we first fix i and k, and maximize the above by changing l and j. Then the above problem is equivalent to maximizing the following for $i = 1, ..., n$ and $k = 0, ..., i - 1$.

$$A_{column}[i, k] = max_{l=0,j=n/2+1}^{n/2-1,n} \{-s[i, l] + s[k, l] + s[i, j] - s[k, j]\}$$

Let $s^*[i,j] = -s[j,i]$. Then the above problem can further be converted into

$$A_{column}[i,k] = -min_{l=0}^{n/2-1}\{s[i,l] + s^*[l,k]\} + max_{j=n/2+1}^{n}\{s[i,j] + s^*[j,k]\}$$

The first part in the above is distance matrix multiplication of the *min* version and the second part is of the *max* version. Let S_1 and S_2 be matrices whose (i,j) elements are $s[i,j-1]$ and $s[i,j+n/2]$. For an arbitrary matrix T, let T^* be that obtained by negating and transposing T. As the range of k is $[0\ ..\ n-1]$ in S_1^* and S_2^*, we shift it to $[1..n]$. Then the above can be computed by multiplying S_1 and S_1^* by the *min* version, multiplying S_2 and S_2^* by the *max* version, subtracting the former from the latter, that is, $S = S_2S_2^* - S_1S_1^*$, and finally taking the maximum from the lower triangle. We will re-organize this maximizing operation into a tournament later. We call the operations of extracting a triangle *triangulation*. This is effectively done by putting $-\infty$ in the upper triangle of S including the diagonal. We call this converted matrix S'.

For simplicity, we assume n is a power of 2. Then all size parameters appearing through recursion in Algorithm M are power of 2. We define the work of computing the three subarrays, A_{column}, $A_{left-row}$. and $A_{right-row}$, to be the work at level 0. The algorithm will split the array horizontally and vertically into four through the recursion to go to level 1.

Now let us analyze the time for the work at level 0. We can multiply $(n, n/2)$ and $(n/2, n)$ matrices by 4 multiplications of size $(n/2, n/2)$, and there are two such multiplications in $S = S_2S_2^* - S_1S_1^*$. We measure the time by the number of comparisons, as the rest is proportional to this. Let $M(n)$ be the time for multiplying two $(n/2, n/2)$ matrices. At level 0, we obtain an A_{column}, $A_{left-row}$, and $A_{right-row}$, spending $12M(n)$ comparisons. Thus we have the following recurrence for the total time $T(n)$. The following lemma [19] is obvious.

$$T(1) = 0,\ T(n) = 4T(n/2) + 12M(n).$$

Lemma 1. *Let c be an arbitrary constant such that $c > 0$. Suppose $M(n)$ satisfies the condition $M(n) \geq (4+c)M(n/2)$. Then the above $T(n)$ satisfies $T(n) \leq 12(1+4/c)M(n)$.*

Clearly the complexity of $O(n^3(\log\log n/\log n)^{1/2})$ for $M(n)$ satisfies the condition of the lemma with some constant $c > 0$. Thus the maximum subarray problem can be solved in $O(n^3(\log\log n/\log n)^{1/2})$ time. Since we take the maximum of several matrices component-wise in line 9 of our algorithm and maximum from S', we need an extra term of $O(n^2)$ in the recurrence to count the number of operations. This term can be absorbed by slightly increasing the constant 12 in front of $M(n)$ in the above recurrence.

5 The *k*-Maximum Subarray Problem

When we solve the maximum subarray problem with Algorithm M, within the same asymptotic time complexity, we organize a four-ary tournament along the

four-way recursion as internal nodes, and the three sub-problems; column centered, left-row centered, and right-row centered as external nodes, in Algorithm M. Those sub-problems are organized into tournaments in the next section. For now we regard them as leaves and assume they can respond to our request in our desired time. When we make the four-ary tournament along the execution of Algorithm M, we copy necessary portions of array a for the seven sub-problems from line 2 to 8. The total overhead time and space requirement for this part are $O(n^2 \log n)$.

Suppose the maximum subarray was returned at level 0, whose coverage and scope are $((K, L), (I, J))$ and $((k, l), (i, j))$. If this array is a single element, that is, returned at the bottom of recursion, i.e., line 1 of the algorithm, we put $-\infty$ at the leaf, and reorganize the tournament for the second maximum subarray towards the root along the winning path. The necessary time is $O(\log n)$.

If the maximum subarray is not from the bottom of recursion, it must be from one of A_{column}, $A_{left-row}$, and $A_{right-row}$ of some coverage at some level. Those three problems are organized into a tournament each, so that they can return the second maximum in $O(\log n)$ time. The coverage and scope information can identify which of the three produced the winner. We can reorganize the tournament along the winning path from this second maximum towards the root. Thus the k-maximum subarray problem can be solved in $O(M(n)+k \log n)$ time, where $M(n) = O(n^3 \sqrt{\log \log n / \log n})$.

Let us assume $K = 1$, $I = n$, $L = 1$, and $J = n$ without loss of generality. Also assume A was obtained from A_{column}, which is in turn obtained from S', that is, the lower triangle of $S = S_2 S_2^* - S_1 S_1^*$. We rewrite this equation as $S = Q - P$, where $P = S_1 S_1^*$ and $Q = S_2 S_2^*$. We assume that $S[i, k]$ for some $k < i$ gives A_{column} with the witnesses l and j for P and Q respectively. We need to find the next value for S'. To do so, we need to find the next minimum value for $P[i, k]$ and next maximum for $Q[i, k]$ with witnesses different from l and j. As is shown in the following sections, the next value for $P[i, k]$ and $Q[i, k]$ are returned in $O(\log n)$ time. Then using the X+Y algorithm, we can choose the next value for $S[i, k]$ in $O(\log n)$ time, which is delivered to the tournament for S' where other elements are intact. Thus the next value for the chosen one of the above three problems, A_{column}, $A_{left-row}$ and $A_{right-row}$, can be found in $O(\log n)$ time.

We observe at this stage that any DMM algorithm, that can deliver successive minimum distances from layer 1 to layer 3 in the context of Section 2 in $O(\log n)$ time, can be fitted into the framework of our algorithm.

6 Distance Matrix Multiplication by Divide and Conquer

We review the DMM algorithm of *min*-version in [18]. The *max*-version is similar. Matrices A, B, and C in DMM are divided into (m, m)-submatrices for $N = n/m$ as follows:

$$\begin{pmatrix} A_{1,1} & \dots & A_{1,N} \\ \dots & & \\ A_{N,1} & \dots & A_{N,N} \end{pmatrix} \begin{pmatrix} B_{1,1} & \dots & B_{1,N} \\ \dots & & \\ B_{N,1} & \dots & B_{N,N} \end{pmatrix} = \begin{pmatrix} C_{1,1} & \dots & C_{1,N} \\ \dots & & \\ C_{N,1} & \dots & C_{N,N} \end{pmatrix}$$

Matrix C can be computed by

$$C = (C_{ij}), \text{ where } C_{ij} = min_{k=1}^{N}\{A_{ik}B_{kj}\}(i, j = 1, ...N). \quad (2)$$

Here the product of submatrices is defined similarly to (1) and the "min" operation is defined on submatrices. Since comparisons and additions of distances are performed in a pair, we measure the time complexity by the number of key comparisons, and omit counting the number of additions for measurement of the time complexity. We have N^3 multiplications of distance matrices in (2). Let us assume that each multiplication of (m, m)-submatrices can be done in $T(m)$ computing time, assuming precomputed tables are available. The time for constructing the tables is reasonable when m is small. The time for min operations in (2) is $O(n^3/m)$ in total. Thus the total time excluding table construction is given by $O(n^3/m + (n/m)^3T(m))$. As shown below, it holds that $T(m) = O(m^2\sqrt{m})$. Thus the time becomes $O(n^3/\sqrt{m})$.

Now we further divide the small (m, m)-submatrices into rectangular matrices in the following way. We rename the matrices A_{ik} and B_{kj} in (2) by A and B. Let $M = m/l$, where $1 \leq l \leq m$. Matrix A is divided into M (m, l)-submatrices A_1, ..., A_M from left to right, and B is divided into M (l, m)-submatrices B_1, ..., B_M from top to bottom. Note that A_k are vertically rectangular and B_k are horizontally rectangular. Then the product $C = AB$ can be given by

$$C = min_{k=1}^{M}C_k, \text{ where } C_k = A_kB_k \quad (3)$$

As shown in the next section, A_kB_k can be computed in $O(l^2m)$ time. Thus the above C in (3) can be computed in $O(m^3/l + lm^2)$ time. Setting $l = \sqrt{m}$ yields $O(m^2\sqrt{m})$ time.

We define a u/l-tournament. Let us find k minima from (n, n)-matrices $X_1, ...,$ X_m for general m and n. The right-hand side of $X = min_{\ell=1}^{m}X_\ell$ is to take minimum values of matrices component-wise. For each (i, j) we organize (i, j) elements of those m matrices into a lower tournament through index ℓ. Then we organize the n^2 roots of those tournaments, which give X, into an upper tournament, We can draw k minima of those matrices from the root of the upper tournament. We call this tournament structure a u/l- tournament.

Now for the extended DMM algorithm, the "min" operation in (2) for each (i, j) is reorganized into a u/l-tournament within the same asymptotic complexity as that of DMM, by the substitution $X_k = A_{ik}B_{kj}$. As C in (2) is regarded as an (N, N)-matrix of (m, m)-matrices, we organize a tournament of N^2 roots of these u/l-tournaments. We note that the matrix C in (3) can be updated by the next minimum in some A_kB_k in $O(M) = O(m/l)$ time by sequential scanning, that is, without a tournament structure

From this construction, we can find the next minimum for the extended DMM in $O(\log n)$ time, since the next minimum in A_kB_k in (3) can be found in $O(1)$ time, as is shown next.

7 How to Multiply Rectangular Matrices

We rename again the matrices A_k and B_k in (3) by A and B. In this section we show how to compute AB, that is,

$$min_{r=1}^{l}\{a_{ir} + b_{rj}\}, \text{ for } i = 1, ..., m; j = 1, ..., m. \quad (4)$$

Note that we do not form tournaments for this "min" operation.

We assume that the lists of length m, $(a_{1r} - a_{1s}, ..., a_{mr} - a_{ms})$, and $(b_{s1} - b_{r1}, ..., b_{sm} - b_{rm})$ are already sorted for all r and s $(1 \leq r < s \leq l)$. The time for sorting will be mentioned later. Let E_{rs} and F_{rs} be the corresponding sorted lists. For each r and s, we merge lists E_{rs} and F_{rs} to form list G_{rs}. In case of a tie, we put an element from E_{rs} first into the merged list. Let H_{rs} be the list of ranks of $a_{ir} - a_{is}$ $(i = 1, ..., m)$ in G_{rs} and L_{rs} be the list of ranks of $b_{sj} - b_{rj}$ $(j = 1, ..., m)$ in G_{rs}. Let $H_{rs}[i]$ and $L_{rs}[j]$ be the ith and jth components of H_{rs} and L_{rs} respectively. Then we have $G_{rs}[H_{rs}[i]] = a_{ir} - a_{is}$ and $G_{rs}[L_{rs}[j]] = b_{sj} - b_{rj}$.

The lists H_{rs} and L_{rs} for all r and s can be made in $O(l^2 m)$ time, when the sorted lists are available. We have the following obvious equivalence for $r < s$.

$$a_{ir} + b_{rj} \leq a_{is} + b_{sj} \Longleftrightarrow a_{ir} - a_{is} \leq b_{sj} - b_{rj} \Longleftrightarrow H_{rs}[i] \leq L_{rs}[j]$$

Fredman [14] observed that the information of ordering for all i, j, r, and s in the rightmost side of the above formula is sufficient to determine the product AB by a precomputed table. This information is essentially packed in the three dimensional space of $H_{rs}[i](i = 1..m; r = 1..l; s = r + 1..l)$, and $L_{rs}[j](j = 1..m; r = 1..l; s = r + 1..l)$. This can be regarded as the three-dimensional packing.

In [18] it is observed that to compute each (i, j) element of AB, it is enough to know the above ordering for all r and s. This can be obtained from a precomputed table, which must be obtained within the total time requirement. This table is regarded as a two-dimensional packing, which allows a larger size of m. leading to a speed-up. In [20] and [21], a method by one-dimensional packing is described.

For simplicity, we omit i from $H_{rs}[i]$ and $L_{rs}[i]$, and define concatenated sequences $H[i]$ and $L[i]$ of length $l(l-1)/2$ by

$$\begin{aligned} H[i] &= H_{1,2} \ldots H_{1,l} H_{2,3} \ldots H_{2,l} \ldots H_{l,l-1} \\ L[i] &= L_{1,2} \ldots L_{1,l} L_{2,3} \ldots L_{2,l}, \ldots L_{l,l-1} \end{aligned} \quad (5)$$

For integer sequence $(x_1, \ldots, x_p)$, let $h(x_1, \ldots, x_p) = x_1 \mu^{p-1} + \ldots + x_{p-1}\mu + x_p$. Let $h(H[i])$ and $h(L[i])$ be encoded integer values for $H[i]$ and $L[i]$, where $p = l(l-1)/2$ and $\mu = 2m$. The computation of h for $H[i]$ and $L[i]$ for all i takes $O(l^2 m)$ time. By consulting a precomputed table *table* with the values of $h(H[i])$ and $h(L[j])$, we can determine the value of r that gives the minimum for (4) in $O(1)$ time. For all i and j, it takes $O(m^2)$ time. Thus the time for one $A_k B_k$ in (3) is $O(\ell^2 m)$, since $l^2 = m$. M such multiplications take $O(M\ell^2 m) = O(\ell m^2)$ time.

To compute $table[x][y]$ for any positive integers x and y, x and y are decoded into sequences H and L, which are expressed by the right-hand sides of (5). If $H_{s,r} > L_{s,r}$ for $s < r$ or $H_{r,s} < L_{r,s}$ for $r < s$, we can say r beats s in the sense that $a_{ir} + b_{rj} \leq a_{is} + b_{sj}$ if H and L represent $H[i]$ and $L[j]$. We first fix r and check this condition for all such s. We repeat this for all r. If r is not beaten by any s, it becomes the table entry, that is, $table[x][y] = r$. If there is no such r, the table entry is undefined. There are $O(((2m)^{l(l-1)/2})^2)$ possible values for all x and y, and one table entry takes $O(l(l-1)/2)$ time. Thus the table can be constructed in $O((l(l-1)/2)(2m)^{2l(l-1)/2}) = O(c^{m \log m})$ time for some constant c. Let us set $m = \log n/(\log c \log\log n)$. Then we can compute the table in $O(n)$ time.

If r is beaten by i participants, the rank of r becomes $i+1$. Let r_i be at rank i. Then we fill the (x, y) entry of $table'$, $table'[x, y]$, by $h(r_1, ..., r_l)$ with $p = l$. That is, using this function h, we encode not only the winner, but second winner, third winner, etc., into the table elements. This can also be done in $O(n)$ time, by a slight increase of constant c in the previous page.

To prepare for the extended DMM, we extend equation (4) in such a way that c_{ij} is the l-tuple of the imaginary sorted sequence, $(a_{ir_1} + b_{r_1 j}, ..., a_{ir_l} + b_{r_l j})$, of the set $\{a_{ir} + b_{rj} | 1 \leq r \leq l\}$. Note that we do not actually sort the set. The leftmost element of c_{ij}, that is, the minimum, participates in the tournament for "min" in (3). If $c_{ij} = (x_1, x_2, ..., x_l)$ and x_1 is chosen as the winner, c_{ij} is changed to $(x_2, ..., x_l, \infty)$, etc. As k can be up to $O(n^3)$, many of c_{ij} will be all infinity towards the end of computation.

This can be implemented by introducing an auxiliary matrix C'. When we compute DMM, we compute C', where $c'_{ij} = table'[h[H[i]), h(L[j])] = h(r_1, ..., r_l)$. Each r_k $(k = 1, ..., l)$ is obtained in $O(1)$ time. The elements of the sorted list of c_{ij} is delivered by decoding $C'[i, j]$ one-by-one when demanded from up-stream of the algorithm.

Example 1. $m = 5$, $2m = 10$, $h(H) = 456$, and $h(L) = 329$. Since $H_{1,2} > L_{1,2}$ and $H_{2,3} < L_{2,3}$, the winner is 2, that is, $table[456, 329] = 2$. Also we see $table'[453, 329] = 213$, since $H[1, 3] > L[1, 3]$.

$$H = \begin{bmatrix} - & 4 & 5 \\ - & - & 6 \\ - & - & - \end{bmatrix}, \quad L = \begin{bmatrix} - & 3 & 2 \\ - & - & 9 \\ - & - & - \end{bmatrix}$$

We note that the time for sorting to obtain the lists E_{rs} and F_{rs} for all k in (3) is $O(Ml^2 m \log m)$. This task of sorting, which we call presort, is done for all A_{ij} and B_{ij} in advance, taking $O((n/m)^2(m/l)l^2 m \log m) = O(n^2 l \log m)$ time, which is absorbed in the main complexity. Thus we can compute k shortest distances in $O(M(n) + k \log n)$ time.

8 Concluding Remarks

We showed an asymptotic improvement on the time complexity of the k-maximum subarray problem based on a fast algorithm for DMM. The time complexity

is sub-cubic in n, when $k = o(n^3/\log n)$. If we use recent faster algorithms for DMM, it may be possible to have a better complexity bound for the k-MSA problem.

Another challenge is to use the same idea of tournament technique for the disjoint k-MSA problem. Once the maximum subarray is found, we need to exclude the occupied portion from further considerations. This was done by "hole creation" in [3], achieving a cubic time for $k = O(n/\log n)$. A "hole" causes many tournaments to be updated to offer the best subarrays to be chosen. It remains to be seen whether a similar technique can be used in the disjoint case to achieve a sub-cubic time for the same range of k.

The authors are very grateful to reviewers, whose constructive comments greatly helped us improve the description of this revised version.

References

1. Bae, S.E., Takaoka, T.: Mesh algorithms for the K maximum subarray problem. In: Proc. ISPAN 2004, pp. 247–253 (2004)
2. Bae, S.E., Takaoka, T.: Improved Algorithms for the K-Maximum Subarray Problem for Small K. In: Wang, L. (ed.) COCOON 2005. LNCS, vol. 3595, pp. 621–631. Springer, Heidelberg (2005) Also in Computer Journal, 49(3), 358–374 (2006)
3. Bae, S.E., Takaoka, T.: Algorithms for K Disjoint Maximum Subarrays. In: Alexandrov, V.N., van Albada, G.D., Sloot, P.M.A., Dongarra, J.J. (eds.) ICCS 2006. LNCS, vol. 3991, pp. 310–339. Springer, Heidelberg (2006) Also in IJFCS 18(2), 310–339 (2007)
4. Bae, S.E.: Sequential and Parallel Algorithms for the Generalized Maximum Subarray Problem, Ph. D Thesis submitted to University of Canterbury (April 2007)
5. Bengtsson, F., Chen, J.: Efficient Algorithms for the k Maximum Sums. In: Fleischer, R., Trippen, G. (eds.) ISAAC 2004. LNCS, vol. 3341, pp. 137–148. Springer, Heidelberg (2004)
6. Bengtsson, F., Chen, J.: A Note on Ranking k Maximum Sums, Technical Report Lulea University LTE-FR-0508 (2005)
7. Bentley, J.: Programming Pearls - Perspective on Performance. Comm. ACM 27, 1087–1092 (1984)
8. Brodal, G.S., Jorgensen, A.G.: A Linear Time Algorithm for the k Maximal Sums Problem, private communication. Also MFCS (to appear, 2007)
9. Chan, T.M.: All pairs shortest paths with real weights in $O(n^3/\log n)$ time. In: Dehne, F., López-Ortiz, A., Sack, J.-R. (eds.) WADS 2005. LNCS, vol. 3608, pp. 318–324. Springer, Heidelberg (2005)
10. Chan, T.M.: More algorithms for all-pairs shortest paths in weighted graphs. In: 39th ACM Symposium on Theory of Computing (STOC), pp. 590–598 (2007)
11. Cheng, C., Cheng, K., Tien, W., Chao, K.: Improved algorithms for the k maximum sums problem. In: Deng, X., Du, D.-Z. (eds.) ISAAC 2005. LNCS, vol. 3827, pp. 799–808. Springer, Heidelberg (2005)
12. Dobosiewicz: A more efficient algorithm for min-plus multiplication. Internt. J. Comput. Math. 32, 49–60 (1990)
13. Frederickson, G.N., Johnson, D.B.: The complexity of selection and ranking in X+Y and matrices with sorted rows and columns. JCSS 24, 197–208 (1982)
14. Fredman, M.: New bounds on the complexity of the shortest path problem. SIAM Jour. Computing 5, 83–89 (1976)

15. Han, Y.: Improved algorithms for all pairs shortest paths. Info. Proc. Lett. 91, 245–250 (2004)
16. Han, Y.: An $O(n^3(\log\log n/\log n)^{5/4})$ time algorithm for all pairs shortest paths. In: Azar, Y., Erlebach, T. (eds.) ESA 2006. LNCS, vol. 4168, pp. 411–417. Springer, Heidelberg (2006)
17. Lin, T.C., Lee, D.T.: Randomized algorithm for the sum selection problem. In: Deng, X., Du, D.-Z. (eds.) ISAAC 2005. LNCS, vol. 3827, pp. 515–523. Springer, Heidelberg (2005)
18. Takaoka, T.: A New Upper Bound on the complexity of the all pairs shortest path problem. Info. Proc. Lett. 43, 195–199 (1992)
19. Takaoka, T.: Sub-cubic algorithms for the maximum subarray problem. In: Proc. Computing:Australasian Theory Symposium (CATS 2002), pp. 189–198 (2002)
20. Takaoka, T.: A Faster Algorithm for the All Pairs Shortest Path Problem and its Application. In: Chwa, K.-Y., Munro, J.I.J. (eds.) COCOON 2004. LNCS, vol. 3106, pp. 278–289. Springer, Heidelberg (2004)
21. Takaoka, T.: An $O(n^3\log\log n/\log n)$ Time Algorithm for the All Pairs Shortest Path Problem. Info. Proc. Lett. 96, 155–161 (2005)
22. Tamaki, H., Tokuyama, T.: Algorithms for the Maximum Subarray Problem Based on Matrix Multiplication. In: Proceedings of the 9th SODA (Symposium on Discrete Algorithms), pp. 446–452 (1998)
23. Zwick, U.: A Slightly Improved Sub-Cubic Algorithm for the All Pairs Shortest Paths Problem. In: Fleischer, R., Trippen, G. (eds.) ISAAC 2004. LNCS, vol. 3341, pp. 921–932. Springer, Heidelberg (2004)

Compressing Spatio-temporal Trajectories

Joachim Gudmundsson[1], Jyrki Katajainen[2,⋆] , Damian Merrick[1,3], Cahya Ong[4], and Thomas Wolle[1]

[1] NICTA⋆⋆, Sydney, Australia
{joachim.gudmundsson,thomas.wolle,damian.merrick}@nicta.com.au
[2] Department of Computing, University of Copenhagen, Denmark
jyrki@diku.dk
[3] School of Information Technologies, University of Sydney, Australia
[4] Department of Engineering, University of New South Wales, Australia

Abstract. Trajectory data is becoming increasingly available and the size of the trajectories is getting larger. In this paper we study the problem of compressing spatio-temporal trajectories such that the most common queries can still be answered approximately after the compression step has taken place. In the process we develop an $O(n \log^k n)$-time implementation of the Douglas-Peucker algorithm in the case when the polygonal path of n vertices given as input is allowed to self-intersect.

1 Introduction

Technological advances in location-aware devices, surveillance systems, and electronic transaction networks are producing more and more opportunities to trace moving individuals. Consequently, an eclectic set of disciplines including geography [7], database research [9], animal-behaviour research [12], and transport analysis [14] shows an increasing interest in movement patterns of various entities moving in various spaces over various times scales (see also the survey by Gudmundsson et al. [8]).

Large sets of data on the movement of entities create the problem of storing, transmitting, and processing this data. Hence, simplifying this data becomes an important problem. Recently, Cao et al. [4] proposed a way of modelling trajectories in 3-dimensional space so that a 3-dimensional path simplification techniques could be applied. Their idea works well in practice and in their experiments the compression rate is in most cases well over 90%. However, their approach has two main drawbacks that we improve on in this paper.

1. They argued that most spatio-temporal queries in databases are composed of the following five types of queries: *where-at*, *when-at*, *intersect*,

⋆ Part of this research was conducted when the author visited Sydney Research Laboratory at NICTA. The research of this author was partially supported by the Danish Natural Science Research Council under contract 272-05-0272 (project "Generic programming—algorithms and tools").

⋆⋆ NICTA is funded through the Australian Government's Backing Australia's Ability initiative, in part through the Australian Research Council.

T. Tokuyama (Ed.): ISAAC 2007, LNCS 4835, pp. 763–775, 2007.

nearest-neighbour and *spatial-join*. However, in their paper they were only able to prove that their approach is "sound" (to be defined) for three of the five query types. In this paper we show that by making a small modification to their model one can prove that all the queries can be approximated.

2. They used the Douglas-Peucker path simplification algorithm which in 3-dimensional space has a running time of $O(n^2)$. In our specific case we show that it can be approximated in $O(n \log^k n)$ time, where k depends on the model.

Simplifying polygonal paths is a well-researched area in cartography, geographic information systems, digital image analysis, and computational geometry. However, trajectories differ from polygonal paths, because trajectories do not only contain information about a sequence of locations, but also *when* an entity has been at these locations. Therefore, simplifying trajectories differs from simplifying polygonal paths, as we might wish to preserve some temporal information. The movement of a point object p is described by a sequence of coordinates given at n time steps $\langle (x_1, y_1, t_1), \ldots, (x_n, y_n, t_n) \rangle$. The aim is to simplify the trajectory such that both spatial and temporal information is maintained.

In this paper we propose an approach that enables us to use 3-dimensional path simplification algorithms that compute a simplified path containing a subset of the vertices of the original path. The computational problem of path simplification is to compute an *optimal* or *minimum ε-simplification*, i.e. an ε-simplification with as few vertices as possible. In applications, this can considerably reduce storage space and processing time.

Imai and Iri [13] formulated the path simplification problem graph theoretically: construct a directed acyclic graph that models all possible edges in a simplification and compute a shortest path in the graph. Their algorithm runs in $O(n^2 \log n)$ time. Chan and Chin [5], and Melkman and O'Rourke [15] improve this running time to quadratic. Most of the known algorithms use $O(n^2)$ time and space. An exception is the algorithm by Agarwal and Varadarajan [1] that achieves $O(n^{4/3+\delta})$ time and space, where $\delta > 0$ is an arbitrarily small constant. However, their algorithm only works for the L_1 metric.

Since the problem of developing a near-linear time algorithm for computing the optimal ε-simplification remains unsolved, several heuristics have been proposed. The most widely used heuristic is the Douglas-Peucker method [6] (and its variants), originally proposed for simplifying curves under the Hausdorff error measure. For a real number $\varepsilon > 0$, the polygonal path $\langle v_1, \ldots, v_n \rangle$ is approximated as follows. If every vertex v_i, for $1 < i < n$, has a distance at most ε to the line ℓ determined by v_1 and v_n, accept the line segment (v_1, v_n) as an approximation for the whole path. Otherwise, split the path at a vertex further than ε from line ℓ and recursively approximate the two pieces. A straightforward implementation requires $O(n)$ time to find the point furthest from line ℓ. Since the recursion depth can be linear, the running time is bounded by $O(n^2)$.

In this paper we show how the algorithm can be implemented more efficiently if we allow the distance computation to be approximate. That is, assume that we are given $\varepsilon > 0$ and that a segment ℓ is about to be tested. Let p be the point furthest from ℓ, and let d be the smallest distance between p and ℓ. We say that

we have an α-approximation, for $\alpha > 1$, if and only if ℓ is accepted if $d \leq \varepsilon$ and discarded if $d > \alpha \cdot \varepsilon$. Note that this implies that ℓ can be either accepted or discarded if $\varepsilon < d \leq \alpha \cdot \varepsilon$.

A crucial aspect of simplification algorithms is how the distance between a point and a line segment is measured. Originally in the Douglas-Peucker algorithm, the Euclidean distance between a point and a line is used (*line model*), where the line is defined by the corresponding line segment. This can lead to counter-intuitive simplifications. That is why we also use the Euclidean distance from a point to a line segment (*line-segment model*). Even though the Douglas-Peucker algorithm does not output the minimum number of vertices and its worst-case running time is $O(n^2)$, it is often used due to its simplicity and efficiency in practice. However, in the case where the path is assumed to be non-self-intersecting, or even monotone, faster methods have been developed. Hershberger and Snoeyink [10] showed that in the line model the running time can be improved in the case where the path does not self-intersect by making use of the fact that the furthest point has to be a vertex of the convex hull of the point set. Allowing $O(n \log n)$ preprocessing they showed how the furthest point can be found in $O(\log n)$ time. This was later improved further to $O(n \log^* n)$ in [11] by the same authors. We will use a similar approach with two crucial differences: the input path may self-intersect, and we consider both the line and the line-segment models. The contribution of this paper is threefold:

1. We consider the problem of simplifying trajectories and modify the model by Cao et al. [4], such that the five types of queries proposed in [4] can be approximated in a sound way (Section 2). As a result it follows that the 3-dimensional path simplifications can be used to compress trajectories.
2. We propose an algorithm that produces an approximate Douglas-Peucker simplification of a trajectory, i.e. a z-monotone path in 3-dimensional space (Section 4). That is, given two real values $\varepsilon > 0$ and $\delta > 0$, the output is a $(1 + \delta)\varepsilon$-simplification. The running time of our algorithm is $O(\frac{1}{\delta^2} n \log^2 n)$ in the line model and $O(\frac{1}{\delta^2} n \log^3 n)$ in the line-segment model. Previously no sub-quadratic time (approximation) algorithm was known.
3. In the process we present an $O(n \log^2 n)$-time (line model) and an $O(n \log^3 n)$-time (line-segment model) implementation of the Douglas-Peucker algorithm in the plane in the case where the polygonal path can self-intersect (Sec. 3).

Due to space constraints, all proofs and figures have been omitted and can be found in the full version.

2 Modelling Trajectories

In this section, we introduce our model for trajectories, which generalises the results in [4]. We give some preliminary definitions e.g. of spatio-temporal queries and soundness of distance functions. Then we describe our model and prove its effectiveness regarding soundness.

2.1 Preliminaries

According to Cao et al. [4] most spatio-temporal queries are composed of the following five types of queries: *where-at*, *when-at*, *intersect*, *nearest-neighbour* and *spatial-join*. We state the semantics of the two most basic queries *where-at* and *when-at* on a trajectory $T = \langle (x_1, y_1, t_1), \ldots, (x_n, y_n, t_n) \rangle$ as follows.

- *where-at*(T, t) returns the location of the entity corresponding to T at time t according to T. If $t < t_1$ or $t > t_n$, then the answer is undefined.
- *when-at*(T, x, y) returns the time t at which a moving object on trajectory T is expected to be at location (x, y). If the location is not on the trajectory, or the moving object visits the location more than once, or is stationary at the location, then the answer is undefined.[1]

Also the notion of soundness of distance functions is discussed in [4]. For a trajectory T, let $q(T)$ denote the answer of some spatio-temporal query q with input T. To make the dependence on both ε and the underlying distance function *dist* explicit, we let a $(dist, \varepsilon)$-*simplification* denote a simplification that is computed using *dist*.

Definition 1. *Let T be a trajectory and T' its $(dist, \varepsilon)$-simplification. The distance function dist is sound for query q, if for each ε there exists a bound δ, such that $|q(T, .) - q(T', .)| \leq \delta$. For the where-at query $|q(T, t) - q(T', t)|$ is the Euclidean distance between the two points given as answers, and for the when-at query $|q(T, x, y) - q(T', x, y)|$ is the difference between the two returned times.*

Cao et al. [4] define distance functions between a point p_m and a line segment $\overline{p_i p_j}$ in 3-dimensional space: E_2 (2-dimensional Euclidean distance), E_3 (3-dimensional Euclidean distance), E_u ($E_u(p_m, \overline{p_i p_j}) = \sqrt{(x_m - x_c)^2 + (y_m - y_c)^2}$ where p_c is the point on $\overline{p_i p_j}$ with $t_m = t_c$) and E_t ($E_t(p_m, \overline{p_i p_j}) = |t_m - t_c|$ where p_c is the point on the 2-dimensional projection of $\overline{p_i p_j}$ onto the xy-plane that is closest to the 2-dimensional projection of p_m onto the xy-plane). They also show that only the distance function E_u is sound for the *where-at* query, and only the distance function E_t is sound for the *when-at* query. Hence, they propose to use a combined distance function based on E_u and E_t which is sound for both queries. This approach combines the strength of both single distances, but also their weaknesses. This combined distance function results in the worst compression ratio among the researched distance functions.

We argue that using E_u gives rise to another problem. Consider a trajectory where an entity moves with high speed along the x-axis (i.e. $y = 0$) and changes slightly its speed. (The effect can be amplified by repeating this pattern.) From a practical point of view we might wish to simplify this trajectory to a line segment, as we are not interested in preserving the marginal speed changes of an entity (e.g. a car) on a long line segment (e.g. a motorway). However, with the E_u distance we are unable to do so.

[1] This definition is taken from [3]. The definition in [4] is similar but considers the stationary case as a special case.

2.2 Our Model

As in [4], we think of a trajectory as a polygonal path in 3-dimensional space. The x- and y-dimensions correspond to the two spatial dimensions in which the entities move. The third dimension is the time t, which enables us to preserve temporal information. If we want to apply a path-simplification algorithm on such a 3-dimensional path, we need a distance measure between points (or lines or line segments) in 3-dimensional space. The two spatial dimensions have the same physical units, but the time dimension has a different unit. We choose to use the Euclidean distance in 3-dimensional space and therefore propose to use a conversion parameter α that transforms time units into space units. Given a point p in 3-dimensional space, the 3-dimensional ball B_p with centre at p and radius ε contains exactly those points within distance at most ε from p. Hence, if we would like to know whether point p' is within distance ε of p, then this is the same as asking whether p' is inside B_p.

In our distance function $dist_\alpha$, the impact of α can be seen in two different ways: either as 'stretching' the t-axis or as 'flattening' the ball B_p. In the former, we can say that the bigger α, the longer the time axis (i.e. the more spatial length units that correspond to one time unit), and always consider a perfect ball B as basis for the distance between two points. In the latter, we keep the coordinate system fixed, but the bigger α the flatter the ball B in the t-dimension. Formally, the distance function is defined as follows.

Definition 2. *The distance $dist_\alpha$ between a point $p_m = (x_m, y_m, t_m)$ and a line segment $\overline{p_i p_j}$ is the shortest Euclidean distance in 3-dimensional space from p_m to a point p_c on $\overline{p_i p_j}$ where 1 time unit is equivalent to α space units, i.e.:*

$$dist_\alpha(p_m, \overline{p_i p_j}) = \sqrt{(x_m - x_c)^2 + (y_m - y_c)^2 + \alpha \cdot (t_m - t_c)^2}$$

The three distance functions E_2, E_3, and E_u defined in [4] are special cases of our distance function, namely $dist_0 \equiv E_2$ (where '$\equiv$' denotes equivalence), $dist_1 \equiv E_3$, and $dist_\infty \equiv E_u$. Choosing $\alpha = 0$ renders the time information irrelevant, and hence it is equivalent to projecting the line segment onto the xy-plane and using the Euclidean distance on it. This distance function has the advantage that it does simplify trajectories but it is not sound for the *where-at* query. The other extreme, $\alpha \to \infty$, denoted as $dist_\infty$, means the ball B_p is flattened into a 2-dimensional disk, which is parallel to the xy-plane. This means that the distance between a point p and a line segment $\overline{p_i p_j}$ is the Euclidean distance between p and p', where p' is the point on $\overline{p_i p_j}$ that has the same time value. This distance function has the advantage that it is sound for the *where-at* query, but it does not simplify trajectories.

Apart from being more general, our approach to be able to choose α has the advantage of allowing any distance function between $dist_0 \equiv E_2$ and $dist_\infty \equiv E_u$. Intuitively, we can fine-tune the trade-off between 'soundness' and 'sensible simplification', and we can prove $dist_\alpha$ to be sound for all α under certain conditions. To make this more precise, we incorporate the speed of entities in our

considerations, where the speed s_ℓ along the line segment ℓ is defined as the distance in the xy-plane divided by the time difference corresponding to ℓ.

Theorem 1. *Let $\ell = \overline{p_i p_j}$ be a line segment with speed s_ℓ that is part of a $(dist_\alpha, \varepsilon)$-simplification of the trajectory $T = \langle p_1, \ldots, p_i, \ldots, p_j, \ldots, p_n \rangle$, and let t be any moment of time with $i \leq t \leq j$. Then we have:*

$$|\textit{where-at}(T,t) - \textit{where-at}(\ell,t)| \;\leq\; \delta_s \;:=\; \frac{\varepsilon}{\sin(\arctan \frac{\alpha}{s_\ell})}$$

The previous theorem tells us that the bigger $\frac{\alpha}{s_\ell}$ becomes the smaller gets δ_s. Hence, the distance function $dist_\alpha$ is sound according to Definition 1 for the *where-at* query for any $\alpha > 0$ as long as $0 < s_\ell < \infty$. However, in practice only a restricted range of values for α might be sensible. For instance setting $\alpha = s_{\max}$, where $s_{\max}$ is the maximum speed along the trajectory, results in $\delta_s \leq \sqrt{2} \cdot \varepsilon$ for the entire trajectory. Also values smaller than $s_{\max}$ might make sense for α in practice. In this case, the slower the speed on a line segment of the simplification is, the smaller δ_s is. In the same way as for the *where-at* query we also obtain that the *when-at* query is sound for $dist_\alpha$, if $\alpha \neq 0$ and $s_\ell > 0$.

Theorem 2. *Let $\ell = \overline{p_i p_j}$ be a line segment with speed s_ℓ that is part of a $(dist_\alpha, \varepsilon)$-simplification of the trajectory $T = \langle p_1, \ldots, p_i, \ldots, p_j, \ldots, p_n \rangle$, and let (x, y) be any point that lies exactly once on both the projections of ℓ and $\langle p_i, \ldots, p_j \rangle$ onto the xy-plane. Then we have:*

$$|\textit{when-at}(\langle p_i, \ldots, p_j \rangle, x, y) - \textit{when-at}(\ell, x, y)| \;\leq\; \delta_t \;:=\; \frac{\varepsilon}{\sin(\arctan \frac{s_\ell}{\alpha})}$$

Hence, the smaller $\frac{s_\ell}{\alpha}$ is, the bigger δ_t is. In practice it is sensible to assume that the speed of entities is bounded from above, it is unreasonable to assume that all entities have a minimum speed; this would forbid an entity to be stationary. Being able to choose α allows a user to fine-tune the trade-off between spatial and temporal soundness of $dist_\alpha$, as reflected by Theorems 1 and 2.

Cao et al. show in [4] that, if a distance function is sound for the *where-at* query, then it is also sound for the *nearest-neighbour* and *intersect* queries, and hence, Theorem 1 carries over to those queries, too. The *spatial-join* is special in the sense that the query itself uses a distance function between trajectories. For $\alpha_1 \leq \alpha_2$ we have that $dist_{\alpha_1}(p_m, \overline{p_i p_j}) \leq dist_{\alpha_2}(p_m, \overline{p_i p_j})$. From results in [4] it then follows that $dist_{\alpha_2}$ is sound for the *spatial-join* that uses the Hausdorff distance function based on $dist_{\alpha_1}$ as distance function between trajectories.

We believe that the definition of the *when-at* query as given in [4] is too strict. When considering the soundness of a distance function, we compare the original trajectory T and its simplification T'. Although all points of T' have a distance to T of at most ε, we could expect that $\textit{when-at}(T, x, y)$ or $\textit{when-at}(T', x, y)$ is undefined for almost all points (x, y), which renders any reasoning about soundness to be difficult. Hence, we propose different semantics for the *when-at* query. As simplified trajectories are approximations anyway, we allow a query region instead of a query point.

– *apx-when-at*(T, x, y, λ) returns a time t at which a moving object on trajectory T is expected to be within distance λ from location (x, y). If there is no such location on the trajectory, then the answer is undefined.

It seems impossible to prove the soundness of this query in the same sense as above. However, we can prove that *apx-when-at* will report a point at time t for which it holds that the entity must have been close to (x, y) at some point in time that is close to t. That is, we can prove a 'soundness' bound that has both a spatial and temporal error. To simplify the statement of the theorem we define *set-apx-when-at*(T, x, y, λ) as reporting the set of time points when the trajectory T is within distance λ from (x, y). For a trajectory T we use $T(t)$ to denote the position in the xy-plane of the entity along T at time t.

Theorem 3. *Let P be a $(dist_\alpha, \varepsilon)$-simplification of trajectory $T = \langle p_1, \ldots, p_n \rangle$. Given a query point $q = (x, y)$ in the xy-plane, let t_1 be the time reported by apx-when-at$(P, x, y, \lambda + \varepsilon)$. There exists a time point t_2 in set-apx-when-at$(T, x, y, \lambda + 2\varepsilon)$ such that $|t_1 - t_2| \leq \varepsilon/\alpha$ and $|T(t_1) - P(t_2)| \leq \varepsilon$.*

Note that if we set α to be greater than the largest speed of the entity then both *where-at* and *apx-when-at* can be sound for small errors at the same time for any input path. This is the first time any such bound has been shown using a single distance function, even though it is approximate in both time and space.

3 A Fast Implementation of the Douglas-Peucker Algorithm for Self-intersecting Polygonal Paths

In this section we present a fast implementation of the Douglas-Peucker algorithm in the case when the polygonal path may self-intersect. We consider two variants of the algorithm, one which works in the line-segment model and another which works in the line model. Hershberger and Snoeyink [11] gave an $O(n \log^* n)$-time algorithm working in the line model when the path does not self-intersect. However, their approach heavily rely on the fact that the path does not self-intersect since additional structure can be used in this case to develop efficient algorithms. Furthermore, their algorithm is developed to work in the line model, not in the line-segment model. In the self-intersecting case, only the trivial $O(n^2)$ time bound is known, to the best of the authors' knowledge.

As a first step, we will prove that, just as in the line model, the furthest point has to be a vertex on the convex hull of the point set (this is the only structural result we were able to reuse from [10,11]). For simplicity we will throughout this section assume that no three points lie on a line.

Lemma 1. *Given a set of $n \geq 3$ points S and a line segment ℓ, the maximum distance between S and ℓ is defined between a vertex p on the convex hull of S.*

An important subproblem that we need to consider is the following.

Problem 1. (Line-segment furthest-point queries (LSFP-queries)) Preprocess an ordered set of n points $p_1, \ldots, p_n$ in convex position in the plane into a data

structure supporting the following query: given a line segment (p_i, p_j), $1 \leq i < j \leq n$, report the point p_k that is furthest from (p_i, p_j) such that $i < k < j$.

Below we will prove that the LSFP-query problem can be transformed into the following problem with only a small loss in time and space complexity.

Problem 2. (Half-plane furthest-point queries (HPFP-queries)) Preprocess a set of n points $p_1, \ldots, p_n$ in convex position in the plane into a data structure supporting the following query: given a point q and a directed line ℓ, report the point p_i that is furthest from q subject to being to the left of ℓ.

Lemma 2. *A set S of n points in convex position in the plane can be preprocessed in $2F(n)+O(n \log n)$ time using $O(n)+S(n)$ space such that LSFP-queries can be answered in $2Q(n) + O(\log n)$ time, where $F(n)$ is the preprocessing time needed to store S in a data structure of size $S(n)$ that answers HPFP-queries in $Q(n)$ time.*

The $O(n \log n)$ time bound in the above lemma comes from the fact that we need to compute the convex hull of S. However, if the points in S are sorted with respect to their x-coordinates in increasing order, then this step can be done in $O(n)$ time. Unfortunately this improvement will not affect the overall time complexity of the Douglas-Peucker algorithm.

3.1 Half-Plane Furthest-Point Queries

The HPFP-query problem was first studied by Aronov et al. [2] and they showed the following two results:

Fact 1. *(Corollary 5 in [2]) There is a data structure that requires $O(n^{1+\beta})$ space and preprocessing time, and supports HPFP-queries in $O(2^{1/\beta} \log n)$ time on n points in convex position, for any real number $\beta > 0$.*

Fact 2. *(Corollary 11 in [2]) There is a data structure that requires $O(n \log^3 n)$ space and polynomial preprocessing time, and supports HPFP-queries in $O(\log n)$ time on n points in convex position.*

We present a data structure that has slightly higher query time, but because of smaller preprocessing time and smaller space consumption our approach leads to a more efficient implementation of the Douglas-Peucker algorithm.

Lemma 3. *One can preprocess a planar set S of n points in convex position in $O(n \log n)$ time using $O(n \log n)$ space such that HPFP-queries on S can be answered in $O(\log^2 n)$ time.*

3.2 Path Simplification in the Line-Segment Model

In this section we merge the results into one single data structure. In particular, we study the problem of preprocessing a polygonal path P with n vertices such that, given a line segment ℓ and a subpath P' of P, the point in P' furthest from ℓ is reported. We will prove the following lemma.

Lemma 4. *A polygonal path $P = \langle v_1, v_2, \ldots, v_n \rangle$ with n vertices in the plane can be preprocessed in time $O(n \log^2 n) + \sum_{i=0}^{\log n} 2^{i+1} F(\frac{n}{2^i}))$, using $O(n \log n) + \sum_{i=0}^{\log n} 2^i S(\frac{n}{2^i})$ space such that, given a line segment ℓ and a subpath $P' = \langle v_i, \ldots, v_j \rangle$ of P, the point in P' furthest from ℓ can be reported in time $O(\log^2 n) + 2 \sum_{i=0}^{\log n} Q(\frac{n}{2^i})$, where $F(n)$ is the preprocessing time needed to construct a data structure of size $S(n)$ that can answer HPFP-queries in $Q(n)$ time.*

The standard Douglas-Peucker algorithm iterates over at most n line segments. Thus, by combining Lemmas 2, 3 and 4 we obtain the following theorem.

Theorem 4. *(line-segment model) For a polygonal path P with n vertices in the plane, the Douglas-Peucker algorithm can be implemented in time $O(n \log^3 n)$ using $O(n \log n)$ space.*

Note that in Lemma 4 presorting could be used to improve the preprocessing time by a logarithmic factor, but this does not have any effect on the asymptotic efficiency of the Douglas-Peucker algorithm.

3.3 Path Simplification in the Line Model

Even if Theorem 4 also holds in the line model, the inclusive structure of the distance queries is not fully utilised. It turns out that path simplification is easier in the line model. Next we show how both the time and the space bounds can be improved by a logarithmic factor. The tools used in this improved construction are basically the same as those used before. The main reason for obtaining this improvement is that a vertex of a convex hull furthest from a line can be reported fast by binary search [16] by determining the two tangents parallel to the given line and returning the furthest of the vertices on these tangents.

The algorithm operates in four steps. First, the vertices on the given polygonal path P of size n are partitioned into canonical sets whose size is a power of 2. Let the collection of these sets be $\mathcal{P} = \{P_1, P_2, \ldots, P_h\}$. The size of P_1 should be the largest power of 2 no greater than n, the size of P_2 the largest power of 2 no greater than $n - |P_1|$, and so on. That is, $h \leq \lceil \log n \rceil$. Second, the canonical sets of $\mathcal{P}$ are presorted according to their x-coordinate. Let $\mathcal{S}$ be the corresponding collection of sorted sets of vertices. Also, associate each vertex in a sorted set with its index in the polygonal path. Third, the convex hulls of the canonical sets are computed. Let the resulting collection be $\mathcal{C}$. Due to presorting, the computation of each convex hull only takes linear time if we use Graham's convex-hull algorithm. Fourth, the recursive subroutine, to be described next, is called with ε, P, $\mathcal{P}$, $\mathcal{S}$, and $\mathcal{C}$.

Assume that the input of the recursive subroutine is real number ε and polygonal path $\langle v_i, v_{i+1}, \ldots, v_k \rangle$ together with the corresponding collections of canonical sets, sorted sets, and convex hulls. The functioning of the recursive subroutine is as follows:

1. Compute the furthest point between the polygonal path and the line ℓ determined by v_i and v_k. This is done by computing the furthest point between

ℓ and the convex hulls, one by one, and by determining the overall furthest point. Let v_j be this vertex.
2. If the distance between line ℓ and vertex v_j is less than or equal to ε, return the line segment (v_i, v_k) as a simplification for P and stop this branch of recursion.
3. Split the path into two subpaths $\langle v_i, v_{i+1}, \ldots, v_j \rangle$ and $\langle v_j, v_{j+1}, \ldots, v_k \rangle$. Correspondingly, split the canonical set containing v_j into smaller canonical sets whose size is a power of 2. This is done by repeatedly halving the canonical set containing v_j until v_j forms a singleton set. For each canonical set created during this process, compute the sorted set of vertices by scanning the sorted set corresponding to the parent canonical set. Finally, compute the convex hulls of the new canonical sets created. After halving a canonical set, it and the corresponding sorted set and convex hull are disposed.
4. Call the recursive routine for both subpaths together with the corresponding collections of canonical sets, sorted sets, and convex hulls.

Let us now analyse the performance of this algorithm for a polygonal path of n vertices. The amount of work done in the three first steps of the main routine is dominated by that required by sorting, i.e. the running time is $O(n \log n)$. In the recursive subroutine in connection with each halving, sorted sets are scanned and convex hulls may be computed, both requiring time linear on the size of the subpaths considered. Since each vertex is involved in $O(\log n)$ halvings, the overall running time of all splits is $O(n \log n)$. At each recursive step, in the furthest-point calculation the number of convex hulls to be considered is bounded by $O(\log n)$ and each distance computation between a line and a convex hull takes $O(\log n)$ time. Naturally, the number of recursive calls is linear in the worst case. Therefore, the total running time of the algorithm is $O(n \log^2 n)$. At any given point in time, each vertex can be in at most one canonical set. Hence, the space bound is $O(n)$. The above discussion can be summarised as follows:

Theorem 5. *(line model) For a polygonal path P with n vertices in the plane, the Douglas-Peucker algorithm can be implemented in time $O(n \log^2 n)$ using $O(n)$ space.*

4 A fast Implementation of the Douglas-Peucker Algorithm in 3-dimensional Space

In this section, we present a fast, approximate version of the Douglas-Peucker algorithm in $\mathbb{R}^3$. The algorithm can be used for 3-dimensional paths that are monotone along the z-axis or for trajectories with two spatial dimensions and one temporal dimension. In addition to taking as input a distance error threshold ε, it takes a real number $\delta > 0$, and produces a simplified path that is within a distance of $(1 + \delta)\varepsilon$ from every vertex of the original path. It is possible to set

$\varepsilon = \frac{\varepsilon^*}{1+\delta}$ to obtain a distance error bound of exactly some desired value ε^*. In this case, δ does not affect the distance threshold, but a larger δ may result in a larger number of vertices in the simplified path. As for the original Douglas-Peucker algorithm, this approach is a heuristic, and we present no bound on the number of vertices.

The general idea of the algorithm is as follows. First, we project the vertices of the original path onto $O(1/\delta^2)$ rotations of the xy-plane, equally spaced in angle around the y- and z-axes, yielding a 2-dimensional projection of the original path that may contain self-intersections. One of the 2-dimensional algorithms of Section 3 is then executed on each of the planes, up to the point where the simplified path is to be split at a vertex. At this point, a split vertex has been chosen for each projection plane, based on the distance in the projection between that vertex and the proposed simplified line segment. From these potential split vertices, take the one with the maximum distance to the line segment over all of the projection planes. Split at this vertex in all planes, and continue executing. We will show that the original distance between the vertex and the line segment in $\mathbb{R}^3$ is at most $(1 + \delta)$ times the maximum projected distance over all of the planes. This property allows us to construct an approximate simplification efficiently in 3-dimensional space.

We start by defining a set Ψ of projection planes. Given two angles $0 \leq \alpha \leq \pi$ and $0 \leq \beta \leq 2\pi$, let $\psi_{\alpha,\beta}$ be the plane obtained by rotating the xy-plane around the y-axis by α radians and around the z-axis by β radians, i.e. the plane with normal vector $\langle \sin\alpha\cos\beta, \sin\alpha\sin\beta, \cos\alpha\rangle$. Suppose we wish to perform $k = \lceil 2\pi/\arccos(1/(1+\delta))\rceil$ discrete rotations around the y-axis, and $2k$ around the z-axis. The angle between successive rotations around either of the axes will be $\theta = \pi/k$. Note that for any real $\delta > 0$, it holds that $0 < \theta < \pi/4$. Now we can define a set of projection planes $\Psi = \{\psi_{i\theta,j\theta} \mid i, j \in \mathbb{Z}, 0 \leq i < (k/2), 0 \leq j < k\}$.

Lemma 5. *Given a plane with normal vector $\hat{\boldsymbol{n}}$, there exists a plane $\psi^* \in \Psi$ with normal vector $\hat{\boldsymbol{n}}^*$ such that the angle between $\hat{\boldsymbol{n}}$ and $\hat{\boldsymbol{n}}^*$ is no more than θ.*

Given a point $p \in \mathbb{R}^3$ and a plane $\psi \in \Psi$, let $proj(p, \psi)$ be the orthogonal projection of p onto the plane ψ, defined as the point of intersection between ψ and the line orthogonal to ψ passing through p. To prove an approximation bound, we first need a bound on the distance between two projected points from their original distance in $\mathbb{R}^3$.

Lemma 6. *Given two points $p, q \in \mathbb{R}^3$, it holds that*

$$|\overline{pq}|\cos\theta \leq \max_{\psi\in\Psi} |\overline{proj(p,\psi)proj(q,\psi)}| \leq |\overline{pq}|$$

In the Douglas-Peucker algorithm, we are not only interested in the distance between two points, but also in the distance between a point and a line. We therefore need to look at the projection of the triangle given by the point and two points on the line.

Lemma 7. *Given three points $p, q, r \in \mathbb{R}^3$ such that $\angle pqr > 2\theta$, it holds that*

$$dist(q, \overline{pr}) \geq \max_{\psi \in \Psi} dist(proj(q, \psi), \overline{proj(p, \psi) proj(r, \psi)}) \geq \frac{dist(q, \overline{pr})}{\sqrt{2 - \cos^2 \theta}}$$

We are now ready for the final result of this section.

Theorem 6. *Given a real number $\delta > 0$, a $(1+\delta)$-approximate Douglas-Peucker simplification can be computed in the line-segment model in $O(\frac{1}{\delta^2} n \log^3 n)$ time using $O(\frac{1}{\delta^2} n \log n)$ space, and in the line model in $O(\frac{1}{\delta^2} n \log^2 n)$ time using $O(\frac{1}{\delta^2} n)$ space.*

References

1. Agarwal, P.K., Varadarajan, K.R.: Efficient algorithms for approximating polygonal chains. Discrete & Computational Geometry 23(2), 273–291 (2000)
2. Aronov, B., Bose, P., Demaine, E.D., Gudmundsson, J., Iacono, J., Langerman, S., Smid, M.: Data structures for halfplane proximity queries and incremental Voronoi diagrams. In: Correa, J.R., Hevia, A., Kiwi, M. (eds.) LATIN 2006. LNCS, vol. 3887, pp. 80–92. Springer, Heidelberg (2006)
3. Cao, H., Wolfson, O., Trajcevski, G.: Spatio-temporal data reduction with deterministic error bounds. In: Proceedings of the 2003 Joint Workshop on Foundations of Mobile Computing, pp. 33–42. ACM Press, New York (2003)
4. Cao, H., Wolfson, O., Trajcevski, G.: Spatio-temporal data reduction with deterministic error bounds. The VLDB Journal 15(3), 211–228 (2006)
5. Chan, W.S., Chin, F.: Approximation of polygonal curves with minimum number of line segments. In: Ibaraki, T., Iwama, K., Yamashita, M., Inagaki, Y., Nishizeki, T. (eds.) ISAAC 1992. LNCS, vol. 650, pp. 378–387. Springer, Heidelberg (1992)
6. Douglas, D.H., Peucker, T.K.: Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. The Canadian Cartographer 10(2), 112–122 (1973)
7. Frank, A.U.: Socio-Economic Units: Their Life and Motion. In: Frank, A.U., Raper, J., Cheylan, J.P. (eds.) Life and motion of socio-economic units. GISDATA, vol. 8, pp. 21–34. Taylor & Francis, London (2001)
8. Gudmundsson, J., Laube, P., Wolle, T.: Encyclopedia of GIS. In: Movement Patterns in Spatio-Temporal Data, Springer (to appear)
9. Güting, R., Boehlen, M.H., Erwig, M., Jensen, C.S., Lorentzos, N., Nardelli, E., Schneider, M., Vazirgiannis, M.: A Foundation for representing and querying moving objects. ACM Transactions on Database Systems 25(1), 1–42 (2005)
10. Hershberger, J., Snoeyink, J.: Speeding up the Douglas-Peucker line-simplification algorithm. In: Proceedings of the 5th International Symposium on Spatial Data Handling, pp. 134–143. IGU Commission on GIS (1992)
11. Hershberger, J., Snoeyink, J.: Cartographic line simplification and polygon CSG formulæ in $O(n \log^* n)$ time. Computational Geometry—Theory and Applications 11(3–4), 175–185 (1998)
12. Hulbert, I.A.R.: GPS and its use in animal telemetry: The next five years. In: Sibbald, A.M., Gordon, I.J. (eds.) Proceedings of the Conference on Tracking Animals with GPS, Aberdeen, UK, pp. 51–60. Macaulay Insitute (2001)

13. Imai, H., Iri, M.: Computational-geometric methods for polygonal approximations of a curve. Computer Vision, Graphics, and Image Processing 36(1), 31–41 (1986)
14. Kwan, M.P.: Interactive geovisualization of activity-travel patterns using three dimensional geographical information systems: A methodological exploration with a large data set. Transportation Research Part C 8(1–6), 185–203 (2000)
15. Melkman, A., O'Rourke, J.: On polygonal chain approximation. In: Computational Morphology, pp. 87–95. North-Holland, Amsterdam (1988)
16. Overmars, M., van Leeuwen, J.: Maintenance of configurations in the plane. Journal of Computer and System Sciences 23(2), 166–204 (1981)

Finding Popular Places

Marc Benkert[1], Bojan Djordjevic[2], Joachim Gudmundsson[2], and Thomas Wolle[2]

[1] Department of Computer Science, Karlsruhe University, Germany
mbenkert@ira.uka.de
[2] NICTA* Sydney, Australia
{joachim.gudmundsson,bojan.djordjevic, thomas.wolle}@nicta.com.au

Abstract. Widespread availability of location aware devices (such as GPS receivers) promotes capture of detailed movement trajectories of people, animals, vehicles and other moving objects, opening new options for a better understanding of the processes involved. We investigate spatio-temporal movement patterns in large tracking data sets. Specifically we study so-called 'popular places', that is, regions that are visited by many entities. We present upper and lower bounds.

1 Introduction

Technological advances of location-aware devices, surveillance systems and electronic transaction networks produce more and more opportunities to trace moving individuals. Consequently, an eclectic set of disciplines including geography, market research, data base research, animal behaviour research, surveillance, security and transport analysis shows an increasing interest in movement patterns of entities moving in various spaces over various times scales [1,7,11]. In the case of moving animals, movement patterns can be viewed as the spatio-temporal expression of behaviours, e.g. in flocking sheep or birds assembling for the seasonal migration. In a transport context, a movement pattern could be a traffic jam.

In this paper we will focus on the problem of computing 'popular places' (also called 'convergence patterns' in [13,14]) among geospatial lifelines. The input is a set E of n moving point objects $\Lambda_1, \ldots, \Lambda_n$ whose locations are known at τ consecutive time-steps $t_1, \ldots, t_\tau$, that is, the trajectory of each object is a polygonal line that can self-intersect, see Fig. 1. For brevity, we will call moving point objects *entities* from now on, and when it is clear from the context, we use Λ to denote an entity or its trajectory. It is assumed that an entity moves between two consecutive time steps on a straight line, and the velocity of an entity along a line segment of the trajectory is constant. Given a set of n entities in the plane, an integer $k > 0$ and a real value $r > 0$, a *popular place* is a square of side length r, that is visited by at least k entities. Throughout the article

* National ICT Australia is funded through the Australian Government's Backing Australia's Ability initiative, in part through the Australian Research Council.

T. Tokuyama (Ed.): ISAAC 2007, LNCS 4835, pp. 776–787, 2007.

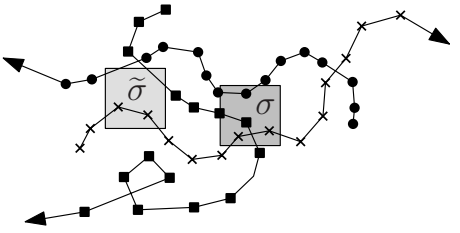


Fig. 1. An example where three entities Λ_1, Λ_2 and Λ_3 are traced during 16 time steps. For $k = 3$ the square $\widetilde{\sigma}$ is a popular place only in the continuous model. While σ is a popular place for $k = 3$ in both the discrete and continuous model.

we will for simplicity assume $r = 1$. Note that the entities do not have to be in the square simultaneously. Spatio-temporal patterns have traditionally been considered in two settings: the discrete case where only the discrete time steps are considered and the continuous case where the polygonal lines connecting the input points are considered. Recently it has been argued [5,8] that the continuous model is becoming more important since trajectories will have to be compressed (simplified) to allow for fast computations. Nowadays it is not unusual that the coordinates are recorded with a frequency of one second. A popular place in the two different models is defined as follows (see Fig. 1).

Definition 1. *Given a set of n moving entities in the plane, an integer $k > 0$ and a real value $r > 0$. An axis aligned square σ of side length r is a* popular place *in the* discrete model *if σ contains input points from at least k different entities. In the* continuous model *σ is a* popular place *if it is intersected by polylines from at least k different entities.*

Recently, there has been considerable research in the area of analysing and modelling spatio-temporal data. In the database community research has mainly focussed on indexing databases so that basic spatio-temporal queries concerning the data can be answered efficiently. Typical queries are spatio-temporal range queries, spatial or temporal nearest neighbours, see for example the work by Săltenis et al. [16] and Hadjieleftheriou et al. [12]. From a data mining perspective Verhein and Chawla [17] used association rule mining to detect patterns in spatio-temporal sets. They defined a region to be a *source, sink* or a *thoroughfare* depending on the number of objects entering, exiting or passing through the region. Mamoulis et al. [15] studied periodic patterns, e.g. yearly migration patterns or daily commuting patterns. There have been several papers considering the problem of detecting flock patterns and leadership patterns [2,3,4,9].

Precursory to this work Laube et al. [13,14] proposed the REMO framework (RElative MOtion) which defines similar behaviour in groups of entities. They defined patterns such as 'flock', 'convergence', 'trend-setting' and 'leadership' based on similar movement properties such as speed, acceleration or movement direction, and gave algorithms to compute them efficiently. They proposed an input model where a ray was drawn from the current position of each entity

that corresponds to its direction. The aim is to find or forecast a popular place (assuming the entities do not change their direction).

As mentioned in earlier work [4,9,10], specifying exactly which of the patterns should be reported is often a subject for discussion. For the discrete model we design a general algorithm that can generate the following output:

- the popular place with the most number of entities (detect maximum),
- a set of rectangles of width 1 and height 2 such that each reported rectangle contains a popular place and all popular places are covered by the reported rectangles (approximate),
- a set of polygons $\mathcal{H}(E)$ such that any axis-aligned unit square with centre in a polygon of $\mathcal{H}(E)$ is a popular place (report all).

In the continuous model we only describe how to find the set $\mathcal{H}(E)$. However, one can easily modify it to any of the output models listed above.

In Section 2 we present an algorithm for the discrete model, followed by an $O(\tau^2 n^2)$ time algorithm in the continuous model. And in Section 4, we present lower bounds and hardness results. We omitted proofs and some details due to space constraints in this extended abstract.

2 A Fast Algorithm in the Discrete Model

A set of n entities is traced over a period of τ time steps, generating τn points in the plane that correspond to the positions of the tracked entities. We will refer to the τn points as *input points*. The input parameter $k > 0$ defines the minimum number of entities defining a popular place, see Definition 1.

The idea of our algorithm is to use a vertical sweep line ℓ sweeping the points from left to right. Together with the sweep line we sweep a vertical strip str_ℓ of width 1 whose right boundary is ℓ. Each of the τn input points induces two event points, one when the point enters str_ℓ and one when the point leaves str_ℓ. We refer to these types as *start* and *end* events.

For a start event, say that an input point p belonging to entity Λ enters str_ℓ, we update our data structures and check for the largest popular place located in str_ℓ that is visited by Λ. Such a popular place must obviously be contained in the axis-aligned rectangle R_p having width 1, height 2 and p on the midpoint

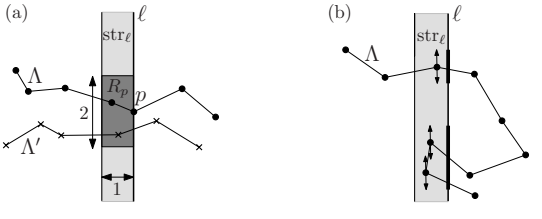


Fig. 2. (a) Finding the popular places in R_p visited by Λ. (b) The set I_Λ is indicated by the bold, solid line segments on ℓ.

of its right vertical segment, see Fig. 2a. If we check at every start event p for a largest popular place within R_p then we will find the maximum popular place. For an end event, say a point p belonging to Λ is about to leave str_ℓ, we simply remove it from the current data structures.

We are going to show how we can maintain a set of trees such that given a y-interval $[a, b]$ (the y-coordinates of the top- and bottom side of R_p), we can find the y-value of the centre of a unit square that contains the largest number of different entities in $O(\log \tau n)$ time per query. Below we will show how we can achieve this by maintaining a tree for each entity separately in $O(\log \tau)$ time per event, and then we show how we can merge this information into one tree T_{int} that can be queried and updated in $O(\log \tau n)$ time.

One Structure for Each Entity. During the sweep, we maintain a set of disjoint y-intervals I_Λ, for each entity Λ, such that I_Λ contains exactly the y-intervals for which a square s in str_ℓ with centre in an interval in I_Λ contains a point of Λ, see Fig. 2b. The square containing a maximum popular place is a square whose centre is contained in a maximum number of such y-intervals.

We will maintain two trees B_Λ and T_Λ for each entity Λ. The tree B_Λ is a balanced binary search tree on the points of Λ currently within str_ℓ and ordered with respect to their y-coordinates. The tree T_Λ will store the set I_Λ of intervals w.r.t. the current position of ℓ. The leaves of T_Λ store the endpoints of the intervals in I_Λ ordered on their y-coordinates. Each leaf also contains a pointer to the leaf in T_Λ containing the other endpoint. Inserting and deleting a new interval can be done in $O(\log \tau)$ time per update.

Assume we are about to process a start event $p_\Lambda = (x, y)$. Let $I = [y - 1/2, y + 1/2]$ and note that any unit square within str_ℓ with centre in I will contain p_Λ. The point is inserted into B_Λ and then a range query is performed in T_Λ that reports the intervals of I_Λ intersecting I. Since the intervals in I_Λ are disjoint and have length at least one, I may intersect at most two intervals in I_Λ. Thus, finding the intersecting intervals can be done in $O(\log \tau)$ time. If the number of intersecting intervals is zero then I is inserted into T_Λ. If I intersects one interval I_1 then I_1 is deleted and the interval $I \cup I_1$ is inserted, and if two intervals I_1 and I_2 are intersected then they are deleted and $I \cup I_1 \cup I_2$ is inserted.

In the case when str_ℓ is about to process an end event $p_\Lambda = (x, y)$, update the trees in a similar manner. Report the two adjacent neighbours $p_1 = (x_1, y_1)$ and $p_2 = (x_2, y_2)$ of p_Λ in B_Λ, and delete p_Λ. Assume without loss of generality that $y_1 < y < y_2$. Let I be as defined above, and let I' be the interval in I_Λ containing I. We have to distinguish three cases:

1. If $|y_2 - y_1| \leq 1$ then we are done since I' does not change.
2. If $\min\{|y - y_1|, |y - y_2|\} > 1$ then $I' = I$ is deleted from T_Λ.
3. If $|y - y_1| > 1$ and $|y - y_2| \leq 1$, or vice versa, then I' is deleted from T_Λ, and the interval $I' \setminus [y - 1/2, y_2 - 1/2)$ is inserted.

We denote the set of all trees T_Λ and B_Λ by $\mathcal{T}^{\text{ent}}$ and $\mathcal{B}^{\text{ent}}$, respectively. Since the total number of events is $2\tau n$ the below corollary follows immediately.

Corollary 1. *Throughout the sweep, the sets* $\mathcal{T}^{\text{ent}}$ *and* $\mathcal{B}^{\text{ent}}$ *can be maintained in* $O(\tau n \log \tau)$ *time requiring* $O(\tau n)$ *space.*

Maintaining the Status of the Sweep. We store all intervals that are currently in str_ℓ in a balanced binary tree T_{int}. We use T_{int} to perform the maximum popular place query for R_p. Let $\mathcal{I} = \bigcup_\Lambda I_\Lambda$. We assume that all start and end points of intervals in $\mathcal{I}$ are pairwise disjoint. The leaf set of T_{int} corresponds to the set of start and end points in $\mathcal{I}$ ordered w.r.t. their y-coordinates.

Since there are τn points, T_{int} contains at most $O(\tau n)$ leaves and thus at most $O(\tau n)$ vertices at all times. During the sweep T_{int} is maintained as follows: every time a tree T_Λ is updated we perform the corresponding update operation in T_{int}. One update in T_Λ requires the deletion and insertion of a constant number of leaves in T_{int}. Thus, one update operation in T_Λ induces update operations in T_{int} that can be performed in $O(\log \tau n)$ time. Since the sweep conducts $2\tau n$ update operations, we have the following:

Lemma 1. *Throughout the sweep, the tree* T_{int} *can be maintained in* $O(\tau n \log \tau n)$ *time requiring* $O(\tau n)$ *space.*

We now show how we can store appropriate information in T_{int} in order to perform maximum popular place queries.

Extending T_{int} to Allow for Efficient Queries. A point $p = (x, y)$ is said to *stab* an interval $[a, b]$ if $y \in [a, b]$. The *stabbing number* of p w.r.t. a set of intervals I is the number of intervals in I that p stabs. Note that for a start event p we have to find the point in $\ell \cap R_p$ having maximum stabbing number w.r.t. $\mathcal{I}$. We maintain this information implicitly in order to do updates as well as queries in $O(\log \tau n)$ time.

For this we store two values, $\text{sum}(V)$ and $\max_{\text{pre}}(V)$, with each vertex V in T_{int}, see Fig. 3. For leaves L we define $\text{sum}(L)$ and $\max_{\text{pre}}(L)$ to be $+1$ if L corresponds to an interval start point and to be -1 otherwise. For inner vertices V let V_{left} and V_{right} denote the left and the right child of V, respectively. Then, sum and $\max_{\text{pre}}$ are defined as follows:

$$\begin{aligned} \text{sum}(V) &:= \text{sum}(V_{left}) + \text{sum}(V_{right}), \quad \text{and} \\ \max_{\text{pre}}(V) &:= \max\{\max_{\text{pre}}(V_{left}), \text{sum}(V_{left}) + \max_{\text{pre}}(V_{right})\} \end{aligned}$$

Let $L_1(V), \ldots, L_m(V)$ be the sequence of all leaves contained in the subtree rooted at V enumerated from left to right. The intuition of the definitions is that $\text{sum}(V) = \sum_{j=1}^{m} \text{sum}(L_j(V))$ and $\max_{\text{pre}}(V) = \max_{1 \le i \le m} \sum_{j=1}^{i} \text{sum}(L_j(V))$.

When performing an update operation in T_{int}, i.e. deleting or inserting a leaf L, updating sum and $\max_{\text{pre}}$ is only required on the path from L to the root. Hence, updating sum and $\max_{\text{pre}}$ takes $O(\log \tau n)$ time per update operation.

Lemma 2. *Throughout the sweep, the values* sum *and* $\max_{\text{pre}}$ *can be maintained in* $O(\tau n \log \tau n)$ *time requiring* $O(\tau n)$ *space.*

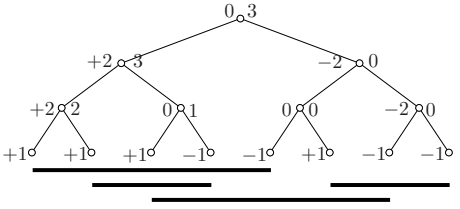


Fig. 3. The values sum (left) and $\max_{\text{pre}}$ (right) for the depicted tree T_{int}

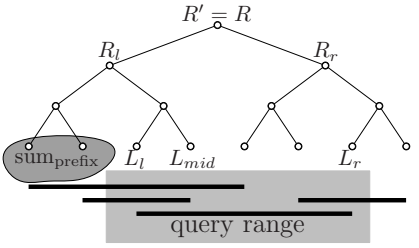


Fig. 4. Querying T_{int} for the maximum stabbing number $\max_{l,r}$

Recall the processing of the sweep. When we arrive at a start event $p = p_\Lambda$, we first perform the required updates in B_Λ and T_Λ, update T_{int} accordingly and then query T_{int} for the most popular place in R_p. Next, we show how the query can be done in $O(\log \tau n)$ time.

Let $L_1, L_2, \ldots$ be the set of leaves in T_{int} ordered from left to right. Each of the leaves in T_{int} is associated with the y-value of the stored start or end point. By performing two searches in T_{int} we can find the leftmost leaf L_l in T_{int} whose y-value is at least $y_p - 1/2$ and the rightmost leaf L_r whose y-value is at most $y_p + 1/2$ in $O(\log \tau n)$ time. This defines our query range within T_{int}: the goal is to find the leaf between (and including) L_l and L_r whose stored point stabs the maximum number of intervals among these leaves. We denote this maximum stabbing number by $\max_{l,r}$. We have that $\max_{l,r} = \max_{l \le m \le r}\{\sum_{j=1}^{m} \text{sum}(L_j)\}$. We claim that $\max_{l,r}$ can be calculated by walking along the search paths from L_l and L_r, respectively, to the root R of T_{int}. We sketch how this is done. Note that the sketch comprises that a leaf with stabbing number $\max_{l,r}$ can be found and reported in $O(\log \tau n)$ time.

Let R' be the least common ancestor of L_l and L_r and let mid, with $l \le mid < r$, be the index such that L_{mid} is the rightmost leaf in the left subtree of R', see Fig. 4. Set $\text{sum}_{\text{prefix}} := 0$, and consider the traversal of the search path from R to L_l. Every time the search path descends into the right subtree of a vertex V add $\text{sum}(V_{left})$ to $\text{sum}_{\text{prefix}}$, see Fig. 4. When the search path reaches L_l we have $\text{sum}_{\text{prefix}} = \sum_{j=1}^{l-1} \text{sum}(L_j)$. Let $\max_l = \max_{l \le m \le mid} \sum_{j=l}^{m} \text{sum}(L_j)$ and $\max_r = \max_{mid+1 \le m \le r} \sum_{j=mid+1}^{m} \text{sum}(L_j)$. Once we know $\max_l$ and $\max_r$, the maximum stabbing number can be computed by $\max_{l,r} = \max\{\text{sum}_{\text{prefix}} + \max_l,\ \text{sum}(R'_{left}) + \max_r\}$. It remains to show how to compute $\max_l$ and $\max_r$.

We compute $\max_l$ by walking along the path from L_l to R'_{left}. Initially, we set $\max_l := \max_{\text{pre}}(L_l)$, and a helper variable $\text{sum}_{\text{seen}} := \text{sum}(L_l)$. Let $L_l = V^0, V^1, \ldots, R'_{left}$ be the sequence of nodes that are traversed when walking from L_l to R'_{left} in T_{int}. Each time we encounter a vertex V^i having V^{i-1} as a left child we look for a new maximum in the right subtree, i.e. $\max_l := \max\{\max_l,\ \text{sum}_{\text{seen}} + \max_{\text{pre}}(V^i_{right})\}$ and update sum_{seen} to $\text{sum}_{\text{seen}} := \text{sum}_{\text{seen}} + \text{sum}(V^i_{right})$. It is not hard to see that in the end of the traversal $\max_l$ holds the right value. There are $O(\log \tau n)$ vertices on the path from L_l to R', and each vertex is processed in constant time. Thus, the time to compute $\max_l$ is $O(\log \tau n)$.

In a similar fashion we compute $\max_r$. Here, we walk along the path from R'_{right} to L_r, let $R'_{right} = V^0, V^1, \ldots, L_r$ be the sequence of traversed vertices. Initially, we set $\max_r := 0$ and $\text{sum}_{\text{seen}} := 0$. Each time we encounter a vertex V_i which is a right child of its parent we look for a new maximum in the left subtree, i.e. $\max_r := \max\{\max_r, \text{sum}_{\text{seen}} + \max_{\text{pre}}(V^{i-1}_{left})\}$ and update sum_{seen} to $\text{sum}_{\text{seen}} := \text{sum}_{\text{seen}} + \text{sum}(V^{i-1}_{left})$. Arriving at L_r we get $\max_r$ by the final update $\max_r = \max\{\max_r, \text{sum}_{\text{seen}} + \max_{\text{pre}}(L_r)\}$. Now, we are ready to summarise the results obtained in this section.

Lemma 3. *Given T_{int} and a y-interval $[l, r]$ one can, in $O(\log \tau n)$ time, return the highest stabbing number $\max_{l,r}$ for any point in $[l, r]$ together with a point $p \in [l, r]$ having this stabbing number.*

Theorem 1. *Given a set E of n moving point objects in the plane the unit square containing the maximum number of different entities in the discrete model can be computed in $O(\tau n \log \tau n)$ time using $O(\tau n)$ space.*

Approximating and Reporting all Popular Places. For a start event $p = p_\Lambda$ we have seen how we can detect a unit square in R_p that contains the maximum number $k_{\max}$ of different entities in the discrete model among all unit squares contained in R_p. If we now find that $k_{\max} \geq k$ we can report R_p as an approximation for (potentially) all popular places that are contained in R_p. This leads directly to the following result.

Theorem 2. *Given a set E of n entities in the plane we can report rectangles of width 1 and height 2 such that each reported rectangle contains a popular place and all popular places are covered by the reported rectangles. This requires $O(\tau n \log \tau n)$ time and $O(\tau n)$ space.*

It gets more involved when we want to report the set of polygons $\mathcal{H}(E)$ such that any axis-aligned unit square with centre in a polygon of $\mathcal{H}(E)$ defines a popular place and each centre of a popular place is contained in $\mathcal{H}(E)$. However, we can extend the above technique in order to show the following theorem.

Theorem 3. *Given a set E of n moving point objects in the plane the polygons $\mathcal{H}(E)$ can be reported in $O(\tau n \log \tau n + M \log \tau n)$ time using $O(\tau n + M)$ space, where M is the number of all popular place defining intervals that we find throughout the algorithm.*

3 An Exact Algorithm in the Continuous Model

In this section we consider the continuous model and present an $O(\tau^2 n^2)$ time algorithm using $O(\tau n)$ space. Later we will argue that it is unlikely to find an asymptotically faster algorithm. Our algorithm takes as input a set E of n entities $\Lambda_1, \ldots, \Lambda_n$ moving in the plane over τ time steps. The output of the algorithm will be a set $\mathcal{H}(E)$ of polygons. $\mathcal{H}(E)$ is the minimal point set in the

plane such that any axis-aligned unit square whose centre lies in $\mathcal{H}(E)$ intersects at least k trajectories of E. Note that these polygons are in general not rectilinear polygons.

We first show how to construct an arrangement $\mathcal{A}$ of lines. The general idea is to sweep the arrangement $\mathcal{A}$ and then building $\mathcal{H}(E)$. For ease of presentation, we will initially describe an algorithm using a standard sweep-line technique with running time $O(\tau^2 n^2 \log \tau n)$ using $O(\tau^2 n^2)$ space. This sweep-line algorithm identifies the edges that contribute to the polygons in $\mathcal{H}(E)$. In a second sweep over these edges, we will construct the polygons in $\mathcal{H}(E)$. Note that the presented algorithms work for inputs with degeneracies and are easy to implement. We then observe that our methods do not require a sweep by a straight line. Hence, we can use a topological plane sweep introduced by Edelsbrunner and Guibas [6] to improve the running time to $O(\tau^2 n^2)$ and the used space to $O(\tau n)$.

Constructing the Line Arrangement. Recall that we use Λ to denote both an entity and its trajectory. Also recall that a trajectory is a polygonal path described by τ points, and that two consecutive points are connected by a straight-line segment s. Consider the following polygon construction: for a trajectory Λ sweep an axis-aligned unit square σ along the trajectory such that its centre moves on Λ as shown in Fig. 5(a). The region swept by σ induces a polygon which we denote by $P(\Lambda)$, see Fig. 5(b). Now consider a set W of polygons and a point q in the plane. The *depth* of q with respect to W is the number of polygons in W intersecting q. This definition allows us to describe $\mathcal{H}(E)$ as follows:

Observation 1. *$\mathcal{H}(E)$ consists of the set of points having depth at least k with respect to $\{P(\Lambda_1), \ldots, P(\Lambda_n)\}$.*

However, we do not explicitly store the polygons $P(\Lambda)$ for each Λ. Instead, the following approach will be used. For each segment s of Λ consider the region swept along s by the unit-square σ. This region is a polygon $P(s)$ with at most six edges, see Fig. 5(c). When considering the two points that specify an edge e_i of $P(s)$, we call the point with the smaller x-coordinate (or smaller y-coordinate, in case the x-coordinates are equal) the *start point* of e_i and the other one the *end point* of e_i. For each edge e_i of $P(s)$, we construct an infinite line l_i that contains e_i. For each line l_i, we store the start and end point of e_i on l_i, and to which side s lies, and Λ. We refer to e_i as a *visible edge* and to the rest of l_i as the *invisible line*. The set of all lines constructed as above yields the line arrangement $\mathcal{A}$, which we will sweep. Note that $\mathcal{A}$ contains $O(\tau n)$ lines.

The First Sweep. The algorithm will sweep the arrangement $\mathcal{A}$ from left to right using a vertical sweep line ℓ. The status structure of the sweep line is stored in a list S of size $O(\tau n)$. For convenience we sometimes use S as an ordered string. It contains the current intersections between ℓ and the visible edges in $\mathcal{A}$ ordered along ℓ from top to bottom. Initially, S is empty. For each intersection between ℓ and a visible edge e, we store a *bracket* br in S with pointers between br, the corresponding edge e and the line corresponding to e. A bracket is formally a tuple $\langle i, type, level, depth \rangle$, where i is the entity number corresponding to the

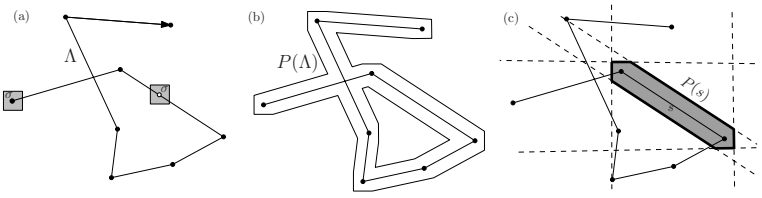


Fig. 5. (a) The square σ that sweeps along a trajectory Λ, (b) the polygon $P(\Lambda)$ obtained from the sweep, (c) the polygon $P(s)$ and the lines l_i generated by $P(s)$ (dashed)

edge; *type* is either *open* or *closed* depending on whether we are entering or exiting the polygon $P(s)$ as we go down ℓ. The brackets will always come in open-closed pairs, because $P(s)$, for a segment s, is a convex region. Note that we can consider them as matching pairs of brackets for the same entity, and that the matching brackets do not have to come from the same polygon $P(s)$. The *depth* value of a bracket is the depth with respect to $\{P(\Lambda_1), \ldots, P(\Lambda_n)\}$ of points immediately after this bracket until the next bracket as we go down ℓ. Note that *depth* only counts each entity once.

An *event* of the sweep is the intersection of exactly two lines of the arrangement $\mathcal{A}$. These events happen at the vertices of the arrangement $\mathcal{A}$, which we call *event points* from now on. By our construction it will happen that three or more lines of $\mathcal{A}$ intersect in one point. Hence, multiple events can occur at a single event point. The moment at which the sweep line reaches an event point x the status of the sweep line is updated according to all events that happen at that event point. All events and event points are computed, sorted and stored beforehand. In the following, we categorise a line l_i that passes through an event point x into: (I) 'x coincides with the end point of e_i', (II) 'x coincides with the start point of e_i' and (III) 'other'. During the sweep, we process all event points in turn and for each event point x we will do the following.

- Let L be the set of lines of $\mathcal{A}$ that go through x. We have $\Theta(|L|^2)$ events.
- Let S' be the substring of S that contains all the brackets that correspond to lines that go through x.
- Let S'' be a new string that contains copies of exactly those brackets contained in S'. From now on, we will modify (brackets of) S''.
- For all pairs of lines l_1 and l_2 in L, let l_1 and l_2 be associated with an edge of $P(s_1)$ and $P(s_2)$ for some segments s_1 and s_2, respectively. We distinguish the following cases, which are illustrated in Fig. 6:
  - (a) Both l_1 and l_2 are of category (II). If $P(s_1) = P(s_2)$ then we encountered a new polygon $P(s)$ and we insert a pair of brackets (anywhere) into S''.
  - (b) Both l_1 and l_2 are of category (I). If $P(s_1) = P(s_2)$ then we finish sweeping a polygon $P(s)$ and delete the corresponding brackets in S''.
  - (c) l_1 is of category (I), l_2 is of category (II). If $P(s_1) = P(s_2)$ then we need to change a pointer. More specifically, let br be the bracket in S''

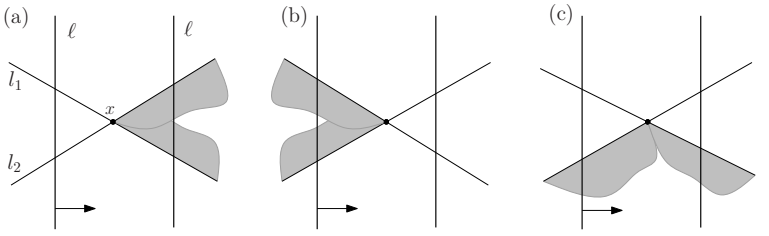


Fig. 6. Cases at an event point x during the line-arrangement sweep

corresponding to the visible segment of l_1. We have to change the pointer of br that points to l_2 so that it now points to l_1.

(*) For all other cases, we do not make any changes.

- We sort the brackets in S'' in non-increasing order according to the slope of their corresponding lines.
- Recalculate all the *level* and *depth* values from scratch for all the brackets in S''. This can be done by looping over S'' using the information in S'.
- Replace S' by S'' in S.

All this can be done in time linear in the number of events at x.

Lemma 4. *The status of the sweep line can be maintained during the sweep in total time $O(\tau^2 n^2)$ and $O(\tau n)$ space.*

Constructing the Output. In the above we did not describe what our algorithm outputs. In this section, we will make up for this, by adding a few steps to the previous sweep. Recall that the output of our algorithm will be a set $\mathcal{H}(E)$ which consists of all points having depth at least k with respect to $\{P(\Lambda_1), \ldots, P(\Lambda_n)\}$.

Whenever we have a bracket br where the two faces above and below the edge e corresponding to br have depth $k-1$ and k then we call that bracket a *boundary bracket*. A boundary bracket br corresponds to a *boundary edge*, which is a part of the edge e that is an edge of a polygon in $\mathcal{H}(E)$. The start and end points of boundary edges are the event points of the sweep where a bracket becomes a boundary bracket, or where a boundary bracket ceases to be a boundary bracket, or where a boundary bracket is inserted, or where a boundary bracket is deleted. To identify the set B of all boundary edges, we extend the procedure of the previous section. We add the following steps to that procedure just before we replace S' by S''.

- For all brackets $br \in S'' \setminus S'$: If br is a boundary bracket, then add a new boundary edge b to B. The start point of b will be the current event point. Associate b with the boundary bracket br.
- For all brackets $br \in S' \setminus S''$: If br is a boundary bracket, then the current event point is the end point of the boundary edge associated with br.

- For all brackets $br \in S' \cap S''$: If br is a boundary bracket in S'' but not in S', then add a new boundary edge b to B. The start point of b will be the current event point. Associate b with the boundary bracket br. If br is a boundary bracket in S' but not in S'', then the current event point is the end point of the boundary edge associated with br. If br is a boundary bracket in S' and in S'', but br corresponds to different lines in S' and S'', then x is a start and end point of boundary edges, which needs to be handled as above.

Having the information about all the boundary edges allows us to build $\mathcal{H}(E)$ by performing a second sweep traversing the set B of boundary edges with a vertical sweep line ℓ. The sweep and the construction of $\mathcal{H}(E)$ can be done in $O(|\mathcal{H}(E)|)$ time, where $|\mathcal{H}(E)|$ denotes the complexity of $\mathcal{H}(E)$. Recall that the first sweep processed $O(\tau^2 n^2)$ events, so together we have:

Theorem 4. *Given a set E of n entities moving in the plane over τ time steps, the set $\mathcal{H}(E)$ can be computed in $O(\tau^2 n^2 \log \tau n)$ time using $O(\tau^2 n^2)$ space.*

Using a Topological Sweep. If we examine the above sweep-line algorithms we can observe that there is no need to process the points strictly from left to right. Also, we only need to keep the events and event points in memory that correspond to the current sweep line. This observation suggests the use of a topological sweep line introduced by Edelsbrunner and Guibas [6]. As a result the above bounds can be improved.

Theorem 5. *Given a set E of n entities moving in the plane over τ time steps, the set $\mathcal{H}(P)$ can be computed in $O(\tau^2 n^2)$ time using $O(\tau n + |\mathcal{H}(P)|)$ space.*

4 Lower Bounds and Hardness Results

We present a lower bound for the popular places problem in the discrete model, and we argue that the popular places problem in the continuous model is at least as hard as 3-SUM, i.e. it is likely that every algorithm in the continuous model of the problem requires quadratic time in the worst case.

Theorem 6. *Let T be a set of n trajectories over τ time-steps, for any $\tau \geq 1$. The problem to decide in the discrete model whether there exists a popular place in T has an $\Omega(\tau n \log \tau n)$ lower bound.*

Theorem 7. *Let T be a set of n trajectories over τ time-steps, for any $\tau \geq 1$. There exists no $o(n^2\tau^2)$ time algorithm to decide whether there exists a popular place in the continuous model with input T, unless there exists an $o(N^2)$ time algorithm to decide* 3-SUM *for an input of total size N.*

Acknowledgements. We would like to thank Hans Bodlaender, Jyrki Katajainen, Damian Merrick and Anh Pham for very useful discussions.

References

1. Wildlife tracking projects with GPS GSM collars (2006), http://www.environmental-studies.de/projects/projects.html
2. Al-Naymat, G., Chawla, S., Gudmundsson, J.: Dimensionality reduction for long duration and complex spatio-temporal queries. In: Proceedings of the 22nd ACM Symposium on Applied Computing, pp. 393–397. ACM, New York (2007)
3. Andersson, M., Gudmundsson, J., Laube, P., Wolle, T.: Reporting leadership patterns among trajectories. In: Proceedings of the 22nd ACM Symposium on Applied Computing, pp. 3–7. ACM, New York (2007)
4. Benkert, M., Gudmundsson, J., Hübner, F., Wolle, T.: Reporting flock patterns. In: Azar, Y., Erlebach, T. (eds.) ESA 2006. LNCS, vol. 4168, pp. 660–671. Springer, Heidelberg (2006)
5. Cao, H., Wolfson, O., Trajcevski, G.: Spatio-temporal data reduction with deterministic error bounds. The VLDB Journal 15(3), 211–228 (2006)
6. Edelsbrunner, H., Guibas, L.J.: Topologically sweeping an arrangement. Journal of Computer and System Sciences 38, 165–194 (1989)
7. Frank, A.U.: Socio-Economic Units: Their Life and Motion. In: Frank, A.U., Raper, J., Cheylan, J.P. (eds.) Life and motion of socio-economic units. GISDATA, vol. 8, pp. 21–34. Taylor & Francis, London (2001)
8. Gudmundsson, J., Katajainen, J., Merrick, D., Ong, C., Wolle, T.: Compressing spatio-temporal trajectories. In: ISAAC (to appear, 2007)
9. Gudmundsson, J., van Kreveld, M.: Computing longest duration flocks in trajectory data. In: Proceedings of the 14th ACM Symposium on Advances in GIS, pp. 35–42 (2006)
10. Gudmundsson, J., van Kreveld, M., Speckmann, B.: Efficient detection of motion patterns in spatio-temporal sets. GeoInformatica 11(2), 195–215 (2007)
11. Güting, R.H., Schneider, M.: Moving Objects Databases. Morgan Kaufmann, San Francisco (2005)
12. Hadjieleftheriou, M., Kollios, G., Tsotras, V.J., Gunopulos, D.: Indexing spatio-temporal archives. The VLDB Journal 15(2), 143–164 (2006)
13. Laube, P., Imfeld, S., Weibel, R.: Discovering relative motion patterns in groups of moving point objects. International Journal of Geographical Information Science 19(6), 639–668 (2005)
14. Laube, P., van Kreveld, M., Imfeld, S.: Finding REMO – detecting relative motion patterns in geospatial lifelines. In: Fisher, P.F. (ed.) Developments in Spatial Data Handling: Proceedings of the 11th International Symposium on Spatial Data Handling, pp. 201–214. Springer, Berlin (2004)
15. Mamoulis, N., Cao, H., Kollios, G., Hadjieleftheriou, M., Tao, Y., Cheung, D.: Mining, indexing, and querying historical spatiotemporal data. In: Proceedings of the 10th ACM SIGKDD International Conference On Knowledge Discovery and Data Mining, pp. 236–245. ACM Press, New York (2004)
16. Săltenis, S., Jensen, C.S., Leutenegger, S.T., Lopez, M.A.: Indexing the positions of continuously moving objects. In: Proceedings of the ACM SIGMOD International Conference on Management of Data, pp. 331–342 (2000)
17. Verhein, F., Chawla, S.: Mining spatio-temporal association rules, sources, sinks, stationary regions and thoroughfares in object mobility databases. In: Lee, M.L., Tan, K.-L., Wuwongse, V. (eds.) DASFAA 2006. LNCS, vol. 3882, pp. 187–201. Springer, Heidelberg (2006)

Maintaining Extremal Points and Its Applications to Deciding Optimal Orientations

Sang Won Bae[1], Chunseok Lee[1], Hee-Kap Ahn[2], Sunghee Choi[1], and Kyung-Yong Chwa[1]

[1] Division of Computer Science, KAIST, Daejeon, Korea
{swbae,stonecold,sunghee,kychwa}@tclab.kaist.ac.kr
[2] Department of Computer Science and Engineering, POSTECH, Pohang, Korea
heekap@postech.ac.kr

Abstract. We consider two non-convex enclosing shapes with the minimum area; the *L-shape* and the *quadrant hull*. This paper proposes efficient algorithms computing each of two shapes enclosing a set of points with the minimum area over all orientations. The algorithms run in time quadratic in the number of given points by efficiently maintaining the set of extremal points.

1 Introduction

Given a set of geometric objects in the plane, there has been a fair amount of work on the smallest enclosing shapes (such as the convex hull, the smallest enclosing disk or the minimum enclosing rectangle) of the objects [2, 10, 11, 12].

In many cases, the enclosing shape is invariant to orientation, that is, the shape does not change over all orientations. Therefore, the enclosing shape for any fixed orientation is the optimal enclosing shape over all orientations (for example, the smallest enclosing circle and the convex hull.) If, however, this is not the case, that is, if the enclosing shape for some fixed orientation changes over different orientations, computing an optimal enclosing shape over all orientations becomes more difficult (for example, the minimum enclosing rectangle of points in the plane.)

Given a set P of points in the plane, we consider two enclosing shapes which are non-convex: the L-shape and the quadrant hull. For a fixed orientation, the *L-shape* can be defined as $R \setminus R'$, where R is an axis-aligned rectangle and R' is a sub-rectangle of R containing the upper right corner of R. Thus, the smallest enclosing L-shape $\mathcal{L}(P)$ of P can be found by taking R as the smallest enclosing rectangle of P and R' as the largest empty sub-rectangle of R containing the upper right corner of R.

The quadrant hull $\mathcal{QH}(P)$ of P is defined as follows: For a fixed orientation, a *quadrant* is the intersection of two half-planes whose supporting lines are axis-aligned and make the right angle. We call a quadrant *free* with respect to P if its interior contains no point in P. Then, the *quadrant hull* $\mathcal{QH}(P)$ of P is

$$\mathcal{QH}(P) := \mathbb{R}^2 - \bigcup_{Q \text{ quadrant free to } P} Q.$$

T. Tokuyama (Ed.): ISAAC 2007, LNCS 4835, pp. 788–799, 2007.

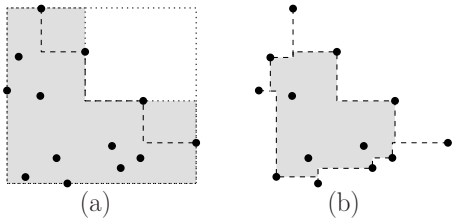


Fig. 1. The minimum enclosing L-shape and the quadrant hull of the same set of points in a fixed orientation

Another equivalent definition of the quadrant hull was suggested by Matoušek [4]. The quadrant hull $\mathcal{QH}(P)$ is also known as *orthogonal convex hull*, which for a fixed orientation can be constructed in time $O(n \log n)$ [7, 8, 9], or faster using integer searching data structures for points with integer coordinates [3].

In this paper, we present efficient algorithms computing a smallest L-shape and quadrant hull of n given points in the plane over all orientations. In doing so, we reveal relations between both enclosing shapes and extremal points. We call a point p in P *extremal* if there is no such point $q \in P$ that $q_x > p_x$ and $q_y > p_y$, where p_x and p_y are the x- and y-coordinates of p in a specific coordinate system.[1] Extremal points form a linear order, for instance, x-coordinate increasing order, and we can build a *staircase* from those ordered points in a natural way (See the dashed segments in Figure 1.) Observe that such a staircase can be described by a sequence of free quadrants supporting two consecutive extremal points.

A minimum enclosing L-shape for a fixed orientation can be obtained by first computing the staircase in the upward and the rightward directions and then picking the best one among the pairs of consecutive points which describe the staircase as in Figure 1(a). On the other hand, the quadrant hull $\mathcal{QH}(P)$ can be described by four staircases as in Figure 1(b). In either case, staircases can be computed in $O(n \log n)$ time, and they allow us in linear time to compute both minimum-area enclosing shapes for a fixed orientation. Therefore, if we could maintain the staircases efficiently while we rotate the coordinate system, it would be helpful to use them in finding an optimal orientation for both enclosing shapes.

Following this motivation, we consider the problem of maintaining the staircase over all orientations in Section 2. We first present a simple, quadratic-time algorithm and then a subquadratic solution with a time/space tradeoff. Sections 3 and 4 are devoted to explaining the algorithms for optimal orientations for both enclosing shapes, which implicitly maintain the staircase and run in quadratic time. Finally, we conclude this paper in Section 5.

2 Maintaining the Staircase

Here, we rotate our coordinate system. We shall denote by *orientation* θ the coordinate system with axes rotated by θ around the origin in counter-clockwise direction. Let P

[1] In this paper, we deal only with orthogonal coordinate systems where two axes make the right angle.

be a set of n points. We denote by $X_\theta(P)$ the set of extremal points of P in orientation θ. Let $\prec_\theta$ be the order of increasing x-coordinate on P in orientation θ. For $X_\theta(P) = \{p_1, \cdots, p_k\} \subset P$, where the indices are given in order $\prec_\theta$, we draw two axis-parallel rays until both meet; one downwards from p_i and the other leftwards from p_{i+1}. Then we get a *step* $s_\theta(p_i, p_{i+1})$ between p_i and p_{i+1} with $1 \leq i < k$ and we denote such a sequence of steps by *the staircase* $S_\theta(P)$ of P in orientation θ. Observe that each quadrant obtained by extending a step is free to P.

Our goal in this section is to maintain $X_\theta(P)$ and $S_\theta(P)$ while θ increases from 0 to 2π. Once the points in $X_\theta(P)$ are given in order $\prec_\theta$, we can easily build $S_\theta(P)$ from $X_\theta(P)$. Therefore, we would like to efficiently update $X_\theta(P)$ to obtain $X_{\theta+\varepsilon}(P)$ for sufficiently small positive ε. To achieve our goal, we focus on when a change between $X_\theta(P)$ and $X_{\theta+\varepsilon}(P)$ occurs. We call an orientation $\varphi \in [0, 2\pi)$ *an event orientation* if $X_{\varphi-\varepsilon}(P) \neq X_{\varphi+\varepsilon}(P)$ for any $\varepsilon > 0$. At each event orientation, an "event" occurs; either a new point in P appears at the staircase or an existing one disappears. We call the former type of events *in-events* and the latter *out-events*.

We use standard data structures. The *event queue* $\mathcal{Q}$ is a priority queue which stores events indexed by their occurring time (or, orientation). We also store the points in $X_\theta(P)$ in a balanced binary search tree $\mathcal{T}$, in order $\prec_\theta$, so that we can add and delete a point in $O(\log n)$ time.

If an in-event occurs at orientation θ, a point $q \in P \setminus X_{\theta-\varepsilon}(P)$ appears to $X_\theta(P)$ and also to S_θ(P). Say that q appears between p and r in $S_\theta(P)$. Then, at orientation θ, the step $s_\theta(q, r)$ between q and r degenerates to a line segment, that is, q lies on $s_\theta(p, r)$. Similarly, when an out-event occurs and q between p and r with $p, q, r \in S_\theta(P)$ is about to disappear, $s_\theta(p, q)$ degenerates to a line segment and q lies on $s_\theta(p, r)$. Figure 2 shows some changes on $S_\theta(P)$ as θ increases, including an in-event and an out-event.

Observation 1. *When an event occurs at orientation θ, there is a degenerate step in $S_\theta(P)$ and one of its two corresponding points is the event point.*

Now, we consider the disk with diameter pr with p and r consecutive in $X_\theta(P)$ with respect to $\prec_\theta$, and its half-disk $D(p, r)$ containing the triangle T_θ defined by $s_\theta(p, r)$ and pr. See Figure 2(a). The corner of $s_\theta(p, r)$ goes along the circular arc of $D(p, r)$ counter-clockwise as θ increases, since T_θ is always a right triangle. We denote two segments of $s_\theta(p, r)$ by $s^h_\theta(p, r)$ and $s^v_\theta(p, r)$, horizontal and vertical ones. Observe that as θ increases $s^h_\theta(p, r)$ sweeps region in $D(p, r)$, and if it encounters a point $q \in P$, we have an in-event and q comes up to the staircase. Among in-events are those corresponding to new extreme points in y-direction; just imagine that we have one more point at infinity in $-x$ direction whose y-coordinate is the same as the point with largest y-coordinate in the current orientation. Also, when $s^h_\theta(p, q)$ degenerates to q we have an out-event and q will disappear from the staircase. See Figure 2.

Observation 2. *Let p and r be two consecutive points in $X_\theta(P)$ with respect to $\prec_\theta$. The next upcoming in-event between p and r occurs by another point q that is first encountered by $s^h_\theta(p, r)$ among points in $D(p, r)$ as θ increases. The out-event of q occurs when $s^h_\theta(p, q)$ degenerates to q.*

By those observations, we know that every event can be captured locally. It is easy in $O(n)$ time to predict the very next in-event between two consecutive extremal points

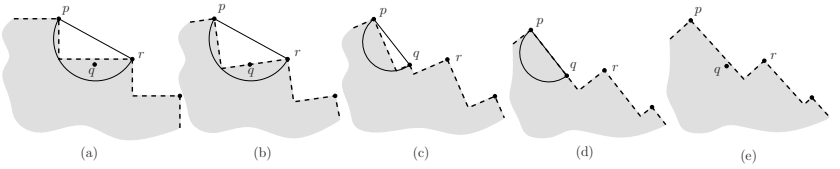


Fig. 2. Changes on $S_\theta(P)$ as θ increases; (a) q lies in $D(p,r)$. (b) $s^h_\theta(p,r)$ hits q, an in-event occurs, and q rises up. (c) $S_\theta(P)$ contains q. (d) $s^h_\theta(p,q)$ degenerates to point q, an out-event occurs, and q is about to disappear. (e) After the out-event. All the other points in P lie in the shaded side of each figure.

with respect to $\prec_\theta$ and the out-event of each extremal point at the current orientation. Indeed, one can reduce this time complexity to predict next events to sublinear by using some complicated data structures, which will be discussed at the end of this section.

Initially, we compute $X_0(P)$ and $S_0(P)$, and store it into $\mathcal{T}$ in $O(n \log n)$ time. Then, we predict in-events and out-events corresponding to $S_0(P)$ and insert them into $\mathcal{Q}$. Now, we are ready to run the main loop. As usual, we extract the upcoming event from the event queue $\mathcal{Q}$, and handle it according to its type. We end this loop if the current orientation θ is at least 2π:

In-event. We put the new point q between p and r into $\mathcal{T}$. Compute the in-events between p and q, and between q and r; also the out-event of q. Insert all the computed events into $\mathcal{Q}$.

Out-event. Remove the disappearing point q between p and r from $\mathcal{T}$, and in-events in which q is involved from $\mathcal{Q}$. Compute the in-event between p and r, and insert it into $\mathcal{Q}$.

The total running time is proportional to the number of events we handle during the algorithm. The following lemma answers the essential question.

Lemma 3. *Any point in P can appear in $X_\theta(P)$ at most four times as θ increases from 0 to 2π.*

Proof. We claim that if a point $q \in X_\theta(P)$ is about to disappear at orientation θ, then q cannot appear again to $X_{\theta+\varphi}(P)$ for $0 < \varphi < \pi/2$. This simply implies the lemma.

We now prove our claim. Assume that $q \in X_\theta(P)$ is about to disappear from $X_\theta(P)$, that is, the out-event of q occurs at θ. Recall by definition that a point $q' \in X_\theta(P)$ if and only if the intersection of two quadrants by extending two steps incident on q' is free to P. (Obviously, the intersection is a quadrant again.) Thus, in orientation θ, observe that there is a point $p \in P$ directly above q since q will disappear after θ (See Figure 2(d).)

Note that maintaining the staircase is invariant under rigid motions. Also, increasing θ is equivalent to rotating the points P clockwise. Now, we transform P by an affine mapping so that q is mapped to the origin. Instead of increasing θ, we rotate p clockwise around q. It is easy to see that p lies in the first quadrant until we rotate the points by more than $\pi/2$. Thus, we conclude our claim and then the lemma. □

Finally, we conclude one of our main results.

Theorem 4. *The staircase has at most $O(n)$ combinatorial changes while rotating the orientation. These changes can be maintained in $O(n^2)$ time and $O(n)$ space.*

2.1 Predicting an In-Event in Sublinear Time

In this subsection, we present a faster way to maintain the staircase. Indeed, the bottleneck of our algorithm is the part of predicting an in-event. Note that predicting an out-event can be preformed in constant time. Here, we present how to predict an in-event in sublinear time so that the total running time of the algorithm reduces to subquadratic, and moreover a time/space tradeoff can be obtained.

Observe that the next in-event between p and q occurs by a point contained in the lune shape $L_\theta(p,q)$ defined by $D(p,q)$ and the half-plane below $s^h_\theta(p,q)$. The first key idea is to restrict candidates to those inside $L_\theta(p,q)$ by the range searching with such a lune shape, which is the intersection of a disk and a half-plane. Since a disk range on the plane can be processed by a half-space range in 3 dimensional space by a well-known lifting-up transformation, the lune shape range searching in $\mathbb{R}^2$ can be viewed as the range searching with intersections of two half-spaces in $\mathbb{R}^3$. Thus we adopt the range searching structure by Matoušek:

Theorem 5 (Matoušek [6]). *Let P be an n-point set in $\mathbb{R}^d$ and let m be a parameter, $n \le m \le n^d$. The range searching problem with the ranges being intersections of p half-spaces, $1 \le p \le d+1$, can be solved with space $O(m)$, query time $O\left(\frac{n}{m^{1/d}} \log^{p-(d-p+1)/d} \frac{m}{n}\right)$, and preprocessing time $O(n^{1+\delta}+m(\log n)^\delta)$, for $\delta > 0$.*

We fix $d = 3$ and $p = 2$ so that we have a structure $\mathcal{R}_m(P)$ for the lune range searching among P. This structure can report all the points in the queried lune shape $L_\theta(p,q)$, but what we need is only one point that will cause the in-event between p and q. In order to get such a point without reporting other points in the range, we make use of special secondary data structures. Note that the range searching structure $\mathcal{R}_m(P)$ is based on a partition tree [1, 5, 6]. Thus, with each internal node v, we associate a secondary structure $\mathcal{C}(P_v)$ for its canonical subset $P_v \subseteq P$. The structure $\mathcal{C}(P_v)$ is supposed to be able to answer a point $r \in P_v$ which we first meet with the line supporting $s^h_\theta(p,q)$ rotated counter-clockwise at q. Such a point r should be in convex position and indeed be the contact point of a tangent line to $\mathrm{conv}(P_v)$ through q. Hence, once we have computed the convex hull $\mathrm{conv}(P_v)$ of P_v, it can be done in time $O(\log |P_v|) = O(\log n)$.

Processing a query $L_\theta(p,q)$, we search $\mathcal{R}_m(P)$ and get at most $O(n/m^{1/3} \log^{4/3} m/n)$ number of canonical subsets. Then, at each corresponding internal node of $\mathcal{R}_m(P)$ we find the same number of candidates for the in-event point. This query process takes $O(n/m^{1/3} \log^{7/3} m/n)$ time. The preprocessing and the storage need an extra factor of $O(\log m) = O(\log n)$, since $\mathcal{R}_m(P)$ has depth at most $O(\log m)$. Thus, we can conclude the following time/space tradeoff result.

Theorem 6. *While rotating the orientation, the staircase of n points can be maintained in $O\left(n^{1+\delta} \log n + m(\log n)^{1+\delta} + \frac{n^2}{m^{1/3}} \log^{7/3} \frac{m}{n}\right)$ time and $O(m \log n)$ storage, where m is a parameter with $n \le m \le n^2$.*

Consequently, if we choose $m = n^{3/2}$, then the above theorem yields an algorithm which runs in $O(n^{3/2}\log^{7/3} n)$ time and $O(n^{3/2}\log n)$ space.

3 Minimum Enclosing L-Shapes

In this section, we present an efficient algorithm for computing an optimal orientation for minimum-area enclosing L-shapes of a set P of n points. First, we observe the following.

Observation 7. *In any orientation, the smallest enclosing L-shape of P touches at least one point in P on each of the six sides of its boundary.*

Thus, we can restrict our candidate L-shapes for each side to have at least one point, and can describe them as these (possibly not disjoint) six points. Note that these points can be shared by two adjacent sides, or possibly by more than two sides when some of them have length zero. Let R_θ be the smallest enclosing rectangle of P in orientation θ and E_θ be the largest empty rectangle which shares the upper right corner with R_θ. Then the smallest enclosing L-shape $\mathcal{L}_\theta(P)$ is represented by $\mathcal{L}_\theta(P) = R_\theta - E_\theta$. Let $p_1, \cdots, p_6$ be six points touching each side of $\mathcal{L}_\theta(P)$; the index is ordered counter-clockwise from the point on the top side of R_θ as shown in Figure 3. Now, we assume p_i's with $1 \le i \le 6$ to represent the smallest L-shapes $\mathcal{L}_\theta(P)$ during orientation $0 \le \alpha < \theta < \beta < 2\pi$; that is, for any $\alpha < \theta < \beta$, the boundary of $\mathcal{L}_\theta(P)$ touches the same sequence of six points in P. The following lemma shows how to get a local optimal orientation over $[\alpha, \beta]$.

Lemma 8. *One can minimize the area of $\mathcal{L}_\theta(P)$ over $0 \le \alpha \le \theta \le \beta < 2\pi$, where we have the same sequence of six points representing $\mathcal{L}_\theta(P)$ in orientation $\theta \in (\alpha, \beta)$.*

Proof. We use all the symbols as defined above. Furthermore, we need more terms. Define $d_1 := \text{length}(p_1p_3)$, $d_2 := \text{length}(p_2p_4)$, $c_1 := \text{length}(p_1p_5)$, and $c_2 := \text{length}(p_4p_6)$. Let $q_1, \cdots, q_6$ be the six corners of $\mathcal{L}_\alpha(P)$ that are ordered by traversing its boundary counter-clockwise from the upper left corner q_1. In orientation α, we consider following angles as in Figure 3: Let $\theta_1 := \angle p_1p_3q_3$, $\theta_2 := \angle p_4p_2q_2$, $\phi_1 := \angle p_5p_1q_6$, and $\phi_2 := \angle p_6p_4q_4$. We then have $\text{area}(R_{\alpha+\delta}) = d_1d_2\sin(\theta_1-\delta)\sin(\theta_2-\delta)$ and $\text{area}(E_{\alpha+\delta}) = c_1c_2\sin(\phi_1+\delta)\sin(\phi_2-\delta)$, where $0 \le \delta \le \beta - \alpha$. Thus, we can express the area of $\mathcal{L}_\theta(P)$ as a function A_L of δ: $A_L(\delta) := \text{area}(\mathcal{L}_{\alpha+\delta}(P)) = \text{area}(R_{\alpha+\delta}) - \text{area}(E_{\alpha+\delta})$. In order to obtain the minimum of A_L on $(0, \beta-\alpha)$, if any, we solve the equation $A'_L(\delta) = 0$, where A'_L is the first derivative of A_L. Through some tedious work on equations, we get $A'_L(\delta) = d_1d_2\sin(2\delta-\theta_1-\theta_2) + c_1c_2\sin(2\delta+\phi_1-\phi_2)$. Solving $A'_L(\delta) = 0$, we have $\delta = \frac{\theta_1+\theta_2}{2} - \frac{1}{2}\arctan\left(\frac{c_1c_2\sin(\theta_1+\theta_2+\phi_1-\phi_2)}{d_1d_2+c_1c_2\cos(\theta_1+\theta_2+\phi_1-\phi_2)}\right)$.

Since $0 < \delta < \beta - \alpha < 2\pi$, the equation $A'_L(\delta) = 0$ has at most a constant number of solutions in the domain, which are either local minima or maxima. A local optimal orientation can be searched among those orientations. □

Now, we are ready to conclude the main theorem of this section.

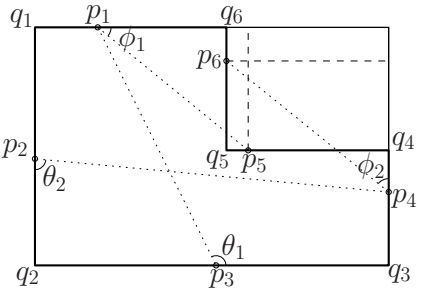


Fig. 3. The smallest L-shape $\mathcal{L}_\alpha(P)$ of P in orientation α with six defining points and other properties

Theorem 9. *Given a set P of n points, one can decide an optimal orientation, which minimizes the area of L-shape $\mathcal{L}_\theta(P)$ enclosing P over $0 \leq \theta < 2\pi$, in $O(n^2)$ time and $O(n)$ space.*

Proof. In this proof, we present an algorithm for computing optimal orientations for L-shapes, which applies the algorithm maintaining the staircase presented in Section 2. Also, we make use of the method described in the proof of Lemma 8. By Lemma 8, if we can fix one point on each side of $\mathcal{L}_\theta(P)$ during $\alpha \leq \theta \leq \beta$ for some $0 \leq \alpha \leq \beta$, then we are able to decide the local optimum in the domain for those at most 6 points.

We need to handle another type of events that occur when the current orientation θ is parallel or perpendicular to an edge of $\mathrm{conv}(P)$. Such an event is easy to predict once we have computed $\mathrm{conv}(P)$. The total number of all events still remains linear in n. Let α_i be the event orientations ordered by their occurrences. For each interval $[\alpha_i, \alpha_{i+1}]$, we have the same four points determining R_θ for every $\theta \in (\alpha_i, \alpha_{i+1})$ and further $X_\theta(P)$ does not change in the interval. Recall that $\mathcal{L}_\theta(P) = R_\theta - E_\theta$ and E_θ is determined by one step of $X_\theta(P)$. Thus, we minimize every L-shape determined by each of step in $X_\theta(P)$ in the interval $[\alpha_i, \alpha_{i+1}]$, and then pick the minimum of these minima. This in $|X_\theta(P)|$ time gives us the local optimal orientation in the orientation interval. Since we have $O(n)$ number of such intervals, all the process can be done in $O(n^2)$ time in order to get a global optimum from gathered local optima. □

Note that the bottleneck of our algorithm is part of computing a local optimum in each step, consuming at most $O(n)$ time. If one could improve this, a subquadratic algorithm will be obtained.

One might wonder if there exists such a nice nature that, for instances, optimal orientations are parallel or perpendicular to an edge of the convex hull of P, or there are at least two points on a segment of the boundary of the L-shape in optimal orientation. If so, one can easily compute the smallest enclosing L-shape over all orientations by testing a small number of candidate orientations. However, L-shapes do not have such a property.

See Figure 4. Let $P := \{(1, \tan\epsilon), (\tan\epsilon, 1), (0, 0)\}$. Then, for $0 < \epsilon < \pi/12$, it is not difficult to check that 0 is the only optimal orientation, which does not come from any pair of points in P. Precisely, there exists such a set P of points that no orientations

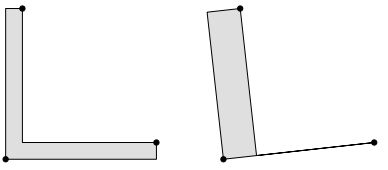


Fig. 4. Three points P and their enclosing L-shapes $\mathcal{L}_0(P)$ and $\mathcal{L}_\epsilon(P)$

parallel or perpendicular to a line through a pair of points in P are optimal for the area of L-shapes.

4 Minimum Quadrant Hulls

In this section, we consider how to find an optimal orientation for quadrant hulls. As aforementioned, we make use of maintaining the staircase described in Section 2 to get an efficient solution. First, recall that the quadrant hull $\mathcal{QH}_\theta(P)$ in orientation θ can be described by four staircases, $S_\theta(P)$, $S_{\theta+\frac{\pi}{2}}(P)$, $S_{\theta+\pi}(P)$, and $S_{\theta+\frac{3}{2}\pi}(P)$. We call two staircases $S_\theta(P)$ and $S_{\theta'}(P)$ *opposite staircases* if θ and θ' differ by π , and *adjacent staircases* if θ and θ' differ by $\pi/2$. We denote the set of the four staircases by $\mathcal{S}_\theta(P)$. Note that $\mathcal{QH}_\theta(P) = \mathcal{QH}_{\pi/2+\theta}(P)$ and $\mathcal{S}_\theta(P) = \mathcal{S}_{\pi/2+\theta}(P)$. We are thus interested only in orientations up to $\pi/2$, and each event orientation $\varphi = \frac{\pi}{2}k + \varphi'$ with $0 \leq \varphi' < \frac{\pi}{2}$ will be processed at φ'.

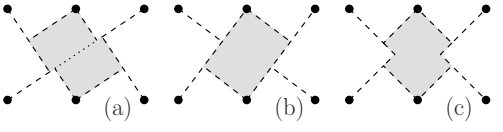


Fig. 5. A set P of 6 points and its quadrant hulls in three orientations

As in the L-shape problem, there seems no hope to restrict the number of candidate orientations for the minimum quadrant hull to a small number. Figure 5 shows a set P of points such that no orientations parallel or perpendicular to a line through a pair of points in P are optimal. Explicitly, $P = \{(-3,-2),(0,-2),(3,-2),(-3,2),(0,2),(3,2)\}$, and Figure 5(a) and (b) show $\mathcal{QH}_{\tan^{-1}\frac{2}{3}}(P)$ and $\mathcal{QH}_{\tan^{-1}\frac{3}{4}}(P)$; these two orientations come from pairs of the points. However, $\mathcal{QH}_{\pi/4}(P)$ as shown in Figure 5(c) is indeed optimal.

We thus devise a solution to the quadrant hull in a similar way as done in L-shapes. Furthermore, we have another difficulty in calculating the area of $\mathcal{QH}_\theta(P)$; as seen in Figure 6, two staircases among those in $\mathcal{S}_\theta(P)$ can cross each other. We observe the following about such overlaps.

Observation 10. *At most one pair of opposite staircases can cross each other. If a step crosses another, it does not cross any other and thus all such overlaps are axis-aligned rectangles.*

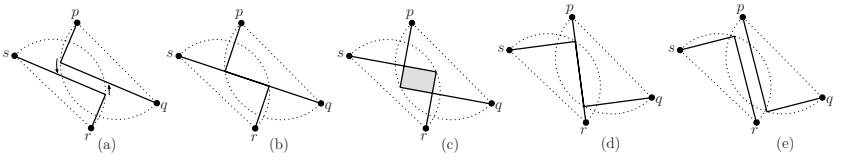


Fig. 6. (a) Two steps in opposite staircases. (b) A step-cross event occurs. (c) The shaded region is the overlap. (d) A step-release event occurs. (e) Two steps are separated.

Considering overlaps, we should refine the definition of combinatorially equivalent quadrant hulls since the shape of $\mathcal{QH}_\theta(P)$ can be changed regardless of the equivalence of $\mathcal{S}_\theta(P)$. Thus we take how $\mathcal{S}_\theta(P)$ crosses also into account, and define *step-cross* and *step-release* events as when two steps in opposite staircases start crossing and when both become free to each other, respectively.

Predicting and handling these events including in-events and out-events is not very difficult if we are allowed $O(n)$ time: For a step s in $\mathcal{S}_\theta(P)$, we check all steps in the opposite staircase to s whether and when each will cross s, and pick the earliest one. We call the resulting event "the step-cross event of a step s". The step-release event of a pair of crossed steps is easy to compute.

In-event. An in-event removes a step but creates two new steps. We perform the same operations as in maintaining the staircase. Further, compute the corresponding step-cross events, if any, and insert them into $\mathcal{Q}$ and delete the useless step-cross event of the removed step from $\mathcal{Q}$.

Out-event. Similarly, compute corresponding step-cross events and delete useless events from $\mathcal{Q}$.

Step-cross event. Compute the step-release event of the involved pair of steps and insert it to $\mathcal{Q}$.

Step-release event. Compute the step-cross event of each of the two involved steps and insert them into $\mathcal{Q}$.

Lemma 11. *We have at most $O(n)$ events in total.*

Proof. In Section 2, we have shown that there are at most $O(n)$ number of in-events and out-events. Here, we show that we have only $O(n)$ step-cross events, which implies the same number of step-release events, and thus at last $O(n)$ events in total.

If we have a step-cross event at θ, we can see two corresponding steps just meeting and a segment joining two points among the four points in the two steps. In Figure 6(b), we can see segment sq as the union of horizontal segments of two steps at the moment of the step-cross event. We call the segment sq *a step-cross segment*. Now, we draw such a step-cross segment whenever we have a step-cross event. We claim that no two step-cross segments cross each other. This implies that we have drawn a plane graph on n points and thus we have at most $O(n)$ such segments.

Let $G = (P, E)$ be a graph whose vertices are P and edges are drawn as explained above. Assume to the contrary that there exist $ab, pq \in E$ such that the two corresponding step-cross segments cross. There should be involved steps, namely, $s_{\theta_1}(b, c)$ and $s_{\theta_1}(d, a)$, and $s_{\theta_2}(q, r)$ and $s_{\theta_2}(s, p)$. Without loss of generality, assume that a is to the

left of b in orientation θ_1. We can find two lines ℓ_1 and ℓ_2 perpendicular to ab and pq, respectively, that separate c and d, and r and s, respectively.

In orientation θ_1, ℓ_1 and the line supporting ab partition the plane into four quadrants. We denote the left upper region by LU_1 and the right lower region by RL_1. $c \in LU_1$ and $d \in RL_1$. Note that there is no point in the right upper and the left lower regions since these two regions are contained in the two quadrants defined by two steps $s_{\theta_1}(b, c)$ and $s_{\theta_1}(d, a)$. Thus, p and q must be separated by ℓ_1 since ab and pq cross. Now, we can assume that $p \in LU_1$ and $q \in RL_1$. Also in orientation θ_2, the line supporting pq and ℓ_2 partition the plane into four regions. Similarly, we define two regions LU_2 and RL_2, and then $r \in LU_2$ and $s \in RL_2$.

Since ab and pq cross and $p \in LU_1$, we know that a lies below pq and b lies above pq. Thus, a must lie in RL_2 and b in LU_2. This implies that a lies to the left of ℓ_1 and to the right of ℓ_2, and b lies to the right of ℓ_1 and to the left of ℓ_2. ℓ_1 and ℓ_2 divide the plane into four cones. We let A be one of the cones where a is located and B be where b is located. The angle of A (or B) at apex $\ell_1 \cap \ell_2$ should be larger than $\pi/2$ since ℓ_1 and ab are perpendicular. However, this is turned to be false: Since $p \in LU_1$ and $q \in RL_1$, p lies to the left of ℓ_1 and of ℓ_2, and q lies to the right of ℓ_1 and of ℓ_2. Since ℓ_2 and pq are also perpendicular, it is required for other cones than A and B to have angle at the apex larger than $\pi/2$, which leads to a contradiction to our assumptions. □

The main loop is performed while $\theta < \pi/2$; since we handle four staircases at the same time, all the required events occur before $\pi/2$. The space of orientations is partitioned into a linear number of intervals $[\alpha_i, \alpha_{i+1})$, where i runs from 0 to the total number of events. The following subsection gives us a way to compute a local optimum in each such interval in linear time.

4.1 Finding Local Optima of $\mathcal{QH}_\theta(P)$

Here, we assume that an orientation interval (α, β) does not contain any event orientation and the current orientation θ runs in between (α, β). Thus, we can also say that $\mathcal{S}_\theta(P)$ contains m points of P and k overlaps. Let p_1 be the first point in $S_\theta(P)$ (or the point with highest y coordinate in orientation θ), and p_i be the i-th point in $\mathcal{S}_\theta(P)$ in clockwise order. We get a polygon $\mathcal{P}$ by a sequence of m sides $p_i p_{i+1}$ with $1 \leq i \leq m-1$, and $p_m p_1$. Observe that two sides cannot cross each other so that we can compute the area of $\mathcal{P}$ in $O(m)$ time. Recall that since we have no event in $\theta \in (\alpha, \beta)$, these extremal points and $\mathcal{P}$ do not change in the interval. Also, we let $s_i(\theta)$ be the step of p_i and p_{i+1} at θ, and $\triangle_i(\theta)$ be the right triangle defined by $s_i(\theta)$ as in Section 2.

Let $\square_j(\theta)$ be the rectangle defined by each overlap in $\mathcal{S}_\theta(P)$, where $1 \leq j \leq k$. Then, the area of $\mathcal{QH}_\theta(P)$ is $\mathrm{area}(\mathcal{QH}_\theta(P)) = \mathrm{area}(\mathcal{P}) - \sum_i \mathrm{area}(\triangle_i(\theta)) + \sum_j \mathrm{area}(\square_j(\theta))$. To minimize the area of $\mathcal{QH}_\theta(P)$ over $\theta \in (\alpha, \beta)$, we analyze the functions $\mathrm{area}(\triangle_i(\theta))$ and $\mathrm{area}(\square_j(\theta))$, and obtain a nice representation so that we can get an optimum in an analytic way.

Lemma 12. *Let (α, β) with $0 \leq \alpha < \beta < \pi/2$ be an orientation interval such that it contains no event orientation. Once we have computed $\mathcal{S}_\alpha(P)$ and $\mathcal{QH}_\alpha(P)$, it is possible in linear time to compute a local optimal orientation in which the area of $\mathcal{QH}_\theta(P)$ is minimized over $\theta \in (\alpha, \beta)$.*

Proof. Let d_i be the length of the hypotenuse of $\triangle_i(\alpha)$ and ϕ_i be the internal angle of $\triangle_i(\alpha)$ at p_i. Then, $\text{area}(\triangle_i(\theta)) = d_i^2 \cos(\phi_i + (\theta - \alpha)) \sin(\phi_i + (\theta - \alpha))$. Substituting $\theta - \alpha$ by φ, we get $\text{area}(\triangle_i(\theta)) = d_i^2 \cos(\phi_i + \varphi) \sin(\phi_i + \varphi) = \frac{1}{2} d_i^2 \sin 2(\phi_i + \varphi) = \frac{1}{2} d_i^2 \sin 2\phi_i \cos 2\varphi + \frac{1}{2} d_i^2 \cos 2\phi_i \sin 2\varphi$, which is linear in $\cos 2\varphi$ and $\sin 2\varphi$. Therefore, the sum over all i is of the same form: $\sum_i \text{area}(\triangle_i(\theta)) = \frac{1}{2} \left(\sum_i d_i^2 \sin 2\phi_i\right) \cos 2\varphi + \frac{1}{2} \left(\sum_i d_i^2 \cos 2\phi_i\right) \sin 2\varphi$.

Now, we deal with each overlap $\square_j(\theta)$ in which two steps $s_a(\theta)$ and $s_b(\theta)$ are involved. Assume without loss of generality that $s_a(\theta)$ belongs to $S_\theta(P)$ and $s_b(\theta)$ to $S_{\pi+\theta}(P)$. If the coordinate of p_i in orientation α is (x_{p_i}, y_{p_i}), $\text{area}(\square_j(\alpha)) = |x_{p_a} - x_{p_b}| \times |y_{p_{a+1}} - y_{p_{b+1}}|$, where $|\cdot|$ returns the absolute value. The point p_i in orientation $\theta = \alpha + \varphi$ can be regarded as the transformed points by rotation by $-\varphi$ in orientation α. Also, the area of $\square_j(\theta)$ is invariant under translations. Thus, $\text{area}(\square_j(\theta)) = \text{area}(\square_j(\alpha+\varphi)) = |x_{p'_a} - x_{p'_b}| \times |y_{p'_{a+1}} - y_{p'_{b+1}}| = |(x_{p_a} - x_{p_b}) \cos\varphi + (y_{p_a} - y_{p_b}) \sin\varphi| \times |(y_{p_{a+1}} - y_{p_{b+1}}) \cos\varphi - (x_{p_{a+1}} - x_{p_{b+1}}) \sin\varphi| = C_1 \cos^2\varphi + C_2 \sin^2\varphi + C_3 \sin\varphi \cos\varphi$, where p'_i is the rotated point of p_i by $-\varphi$ with $\varphi < \pi/2$, and C_1, C_2, and C_3 are constants. This equation can be reformed to be linear in $\sin 2\varphi$ and $\cos 2\varphi$ again $\text{area}(\square_j(\theta)) = C'_1 + C'_2 \cos 2\varphi + C'_3 \sin 2\varphi$, where $C'_1 = \frac{C_1 + C_2}{2}$, $C'_2 = \frac{C_1 - C_2}{2}$, and $C'_3 = \frac{C_3}{2}$.

Consequently, the area of $\mathcal{QH}_\theta(P)$ with $\alpha < \theta < \beta$ can be represented by a function f of φ which is linear in $\sin 2\varphi$ and $\cos 2\varphi$, where $0 < \varphi < \beta - \alpha$. By solving $f'(\varphi) = 0$, where f' is the derivative of f, we get a local minimum (or maximum) of the area of $\mathcal{QH}_\theta(P)$ with $\theta \in (\alpha, \beta)$. All this process is sufficiently done in linear time. □

Now, we conclude the following theorem.

Theorem 13. *Given a set P of n points, one can decide an optimal orientation, which minimizes the area of the quadrant hull $\mathcal{QH}_\theta(P)$ over $0 \le \theta < 2\pi$, in $O(n^2)$ time and $O(n)$ space.*

5 Concluding Remarks

We presented algorithms for computing the smallest enclosing L-shape and quadrant hull of given points for arbitrary orientation. Our solutions for finding smallest enclosing shapes internally maintain extremal points, constituting the staircase, over all orientations. Indeed, maintaining extremal points has its own interest. In computational geometry, dynamic data structures have been intensively researched over last several decades; they mainly focus on how to efficiently handle online updates of their underlying data. Our problem of maintaining the staircase deals with predictable updates over bounded domain, and was proven to be useful in solving geometric optimization problems which are variant to orientations. We carefully expect that many such optimization problems could be efficiently solved by maintaining a certain data structure over all orientations.

References

[1] Agarwal, P.K., Erickson, J.: Geometric range searching and its relatives. In: Chazelle, B., Goodman, J.E., Pollack, R. (eds.) Advances in Discrete and Computational Geometry. Contemporary Mathematics, vol. 233, pp. 1–56. American Mathematical Society Press, Providence (1999)
[2] de Berg, M., van Kreveld, M., Overmars, M., Schwarzkopf, O.: Computational Geometry: Algorithms and Applications, 2nd edn. Springer, Heidelberg (2000)
[3] Karlsson, R.G., Overmars, M.H.: Scanline algorithms on a grid. BIT Numerical Mathematics 28(2), 227–241 (1988)
[4] Matoušek, J., Plecháč, P.: On functional separately convex hulls. Discrete Comput. Geom. 19, 105–130 (1998)
[5] Matoušek, J.: Efficient partition trees. Discrete Comput. Geom. 8, 315–334 (1992)
[6] Matoušek, J.: Range searching with efficient hierarchical cuttings. Discrete Comput. Geom. 10, 157–182 (1993)
[7] Montuno, D.Y., Fournier, A.: Finding the x − y convex hull of a set of x − y polygons. Technical Report 148, University of Toronto (1982)
[8] Nicholl, T.M., Lee, D.T., Liao, Y.Z., Wong, C.K.: On the $X - Y$ convex hull of a set of $X - Y$ polygons. BIT Numerical Mathematics 23(4), 456–471 (1983)
[9] Ottman, T., Soisalon-Soisinen, E., Wood, D.: On the definition and computation of rectilinear convex hulls. Information Sciences 33, 157–171 (1984)
[10] Preparata, F.P., Hong, S.J.: Convex hulls of finite sets of points in two and three dimensions. Communications of the ACM 20(2), 87–93 (1977)
[11] Schwarzkopf, O., Fuchs, U., Rote, G., Welzl, E.: Approximation of convex figures by pairs of rectangles. In: Proc. 7th Ann. Symp. on Theoretical Aspects of Computer Science, pp. 240–249 (1990)
[12] Welzl, E.: Smallest enclosing disks (balls and ellipsoids). In: Maurer, H.A. (ed.) New Results and New Trends in Computer Science. LNCS, vol. 555, pp. 359–370. Springer, Heidelberg (1991)

The Monomial Ideal Membership Problem and Polynomial Identity Testing

V. Arvind and Partha Mukhopadhyay

Institute of Mathematical Sciences
C.I.T Campus,Chennai 600 113, India
{arvind,partham}@imsc.res.in

Abstract. Given a monomial ideal $I = \langle m_1, m_2, \cdots, m_k \rangle$, where m_i are monomials, and a polynomial f as an arithmetic circuit the *monomial Ideal Membership Problem* is to test if $f \in I$. We show that the problem has a randomized polynomial time algorithm for constant k. Furthermore, if k is constant and f is computed by a $\Sigma\Pi\Sigma$ circuit with output gate of *bounded fanin* then we can test whether $f \in I$ in *deterministic* polynomial time.

1 Introduction

Let $\mathbb{F}[x_1, x_2, \cdots, x_n]$ be the ring of polynomials over a field $\mathbb{F}$ with indeterminates $x_1, x_2, \cdots, x_n$. Let $I \subseteq \mathbb{F}[x_1, x_2, \cdots, x_n]$ be an ideal given by a finite generator set $\{g_1, g_2, \cdots, g_r\}$ of polynomials. Then $I = \{\sum_{i=1}^{r} a_i g_i \mid a_i \in \mathbb{F}[x_1, x_2, \cdots, x_n]\}$, and we write $I = \langle g_1, g_2, \cdots, g_r \rangle$. Given an ideal $I = \langle g_1, g_2, \cdots, g_r \rangle$ and a polynomial $f \in \mathbb{F}[x_1, x_2, \cdots, x_n]$ the *Ideal Membership* problem is to decide if $f \in I$.

Ideal Membership Testing is a fundamental algorithmic problem with important applications [Cox92]. In general, however, Ideal Membership Testing is notoriously intractable. The results of Mayr and Meyer show that it is EXPSPACE-complete [MM82, Mayr89]. Nevertheless, because of its important applications, algorithms for this problem are widely studied, mainly based on the theory of Gröbner bases [Cox92].

Polynomial Identity Testing (PIT) is a well-known problem in the field of computational complexity and randomization: given an arithmetic circuit C computing a polynomial in $\mathbb{F}[x_1, x_2, \cdots, x_n]$, the problem is to determine whether the polynomial computed by C is identically zero. Over the years, the question whether PIT is in P has emerged as an important open problem (see, for example, [AB03, KI03]).

In this paper we show interesting connections between Monomial Ideal Membership and Polynomial Identity Testing. Monomial ideals are central to Gröbner basis theory [Cox92]. Suppose $I = \langle m_1, m_2, \cdots, m_k \rangle$ is a monomial ideal in $\mathbb{F}[x_1, x_2, \cdots, x_n]$ generated by the monomials m_i. If $f \in \mathbb{F}[x_1, x_2, \cdots, x_n]$ is given explicitly as an $\mathbb{F}$-linear combination of monomials, then testing membership in the monomial ideal I is trivial: we only need to check if each monomial occurring in f is divisible by some generator monomial m_i. However, as we show in this paper, the problem becomes interesting when f is given by an arithmetic circuit. Given a monomial ideal I in $\mathbb{F}[x_1, \cdots, x_n]$ and an arithmetic circuit C over $\mathbb{F}$ defining a polynomial $f \in \mathbb{F}[x_1, x_2, \cdots, x_n]$, the

T. Tokuyama (Ed.): ISAAC 2007, LNCS 4835, pp. 800–811, 2007.

Monomial Ideal Membership problem is to decide if $f \in I$.[1] It turns out that the problem is still tractable for *constant* k, and its complexity is similar to that of polynomial identity testing. We give a randomized test for Monomial Ideal Membership when f given by an arithmetic circuit and $I = \langle m_1, m_2, \cdots, m_k \rangle$ for constant k. This is analogous to the Schwartz-Zippel polynomial identity test [Sch80, Zip79]. We give a similar randomized test for f given by a black-box when f has small degree. When k is unrestricted we show the problem is coNP-hard but it is in the counting hierarchy.

We study different versions of the problem by placing restrictions on the arithmetic circuit C and the number of monomials generating the ideal I. The identity testing problem for $\Sigma\Pi\Sigma$ circuits has recently attracted a lot of research [DS05, KS07]. The main open problem is whether there is a deterministic polynomial-time identity test for $\Sigma\Pi\Sigma$ circuits. For $\Sigma\Pi\Sigma$ circuits with bounded fanin output gate Kayal and Saxena [KS07] recently gave an ingenious deterministic polynomial-time test. We consider Monomial Ideal Membership, where f is computed by a $\Sigma\Pi\Sigma$ circuit with bounded fanin output gate, and $I = \langle m_1, m_2, \cdots, m_k \rangle$ for constant k. Using ideas from [KS07] we give a *deterministic* polynomial-time algorithm for this problem. More interestingly, we develop the algorithm and its correctness proof based on Gröbner basis theory. As a byproduct, this gives us a different understanding of the identity testing algorithm of [KS07]. We also consider Ideal Membership testing for ideals $I = \langle f_1, \cdots, f_\ell \rangle$ when $f_i \in \mathbb{F}[x_1, \cdots, x_k]$ are arbitrary polynomials but the number of variables k is a constant and show similar upper-bound results. We omit some of the proofs due to lack of space.

2 Preliminaries

We now explain the rudiments of Gröbner basis theory [Cox92, Su98]. Let $\bar{x}$ denote indeterminates $\{x_1, x_2, \cdots, x_n\}$. Let $\mathbb{F}[\bar{x}]$ denotes the polynomial ring $\mathbb{F}[x_1, x_2, \cdots, x_n]$. Let R be a commutative ring. A subring $I \subseteq R$ is an *ideal* of R if $IR \subseteq R$. The Hilbert basis theorem [Cox92] states that any ideal I of $\mathbb{F}[x_1, x_2, \cdots, x_n]$ is *finitely generated*. I.e. we can express $I = \{\sum_{i=1}^{r} p_i g_i \mid p_i \in \mathbb{F}[x_1, x_2, \cdots, x_n]\}$, where the finite collection of polynomials $\{g_1, g_2, \cdots, g_r\}$ is a generating set (or basis) for I. Gröbner bases are defined w.r.t. monomial orderings. We consider only the *lexicographic monomial ordering*. For $\bar{\alpha} = (\alpha_1, \alpha_2, \cdots, \alpha_n) \in \mathbb{N}^n$, let $\bar{x}^{\bar{\alpha}}$ denote the monomial $x_1^{\alpha_1} x_2^{\alpha_2} \cdots x_n^{\alpha_n}$.

Definition 1. *Let* $\bar{\alpha} = (\alpha_1, \alpha_2, \cdots, \alpha_n)$ *and* $\bar{\beta} = (\beta_1, \beta_2, \cdots, \beta_n) \in \mathbb{N}^n$. *We say* $\bar{\alpha} > \bar{\beta}$ *if, in the vector difference* $\bar{\alpha} - \bar{\beta} \in \mathbb{N}^n$, *the left-most nonzero entry is positive. We say,* $\bar{x}^{\bar{\alpha}} > \bar{x}^{\bar{\beta}}$ *(equivalently,* $\bar{x}^{\bar{\beta}} < \bar{x}^{\bar{\alpha}}$*) if* $\bar{\alpha} > \bar{\beta}$.

The monomial ordering fixes a leading monomial $LM(f)$ for a given polynomial f. Let $LC(f)$ denote the coefficient of $LM(f)$. The *leading term* of f is $LT(f) = LC(f)LM(f)$. We now state the general form of the division algorithm over $\mathbb{F}[x_1, x_2, \cdots, x_n]$.

Theorem 1 (Theorem 3, pp.61). [Cox92] *Let* $f \in \mathbb{F}[\bar{x}]$ *and* $(f_1, f_2, \cdots, f_s)$ *be an ordered* s*-tuple of polynomials in* $\mathbb{F}[\bar{x}]$. *Then* f *can be written as,* $f = a_1 f_1 + a_2 f_2 +$

[1] Whenever an ideal is given by a generating set of polynomials, we assume that the exponent of any variable appearing in a generator is in *unary*.

$\cdots + a_s f_s + r$, where $a_i, r \in \mathbb{F}[\bar{x}]$, and either $r = 0$ or r is an $\mathbb{F}$-linear combination of monomials, none of which is divisible by any of $LT(f_1), LT(f_2), \cdots, LT(f_s)$.

We give a brief outline of the proof. Let $\bar{f}$ denote $(f_1, f_2, \cdots, f_s)$. The proof describes a division algorithm Divide$(f; \bar{f})$ which first sorts f by the monomial ordering. The algorithm iteratively tries to eliminate the leading monomial in the current remainder by attempting to divide it with the f_i's in the given order. The f_i that succeeds is the first one whose leading monomial divides the leading monomial of the current remainder. Finally, the remainder r that survives has the property claimed in the theorem. The algorithm terminates since the monomial ordering is a well ordering. The following time bound for Divide$(f; \bar{f})$ is easy to obtain.

Fact 2 (Section 6, pp.12-5). [Su98] *The running time of Divide$(f; \bar{f})$ is $O(s \prod_{i=1}^{n} (d_i + 1)^{O(1)})$, where d_i is the maximum degree of x_i among $f, f_1, f_2, \cdots, f_s$.*

Clearly, $f \in \langle f_1, \cdots, f_s \rangle$ if the remainder $r = 0$ is output by Divide$(f; \bar{f})$. However, the converse is not true in general. Divide$(f; \bar{f})$ can give a nonzero remainder for $f \in \langle f_1, \cdots, f_s \rangle$. In order to make this an ideal membership test, we define *Gröbner bases*. Fix $<$ as the monomial ordering, and let $J \subseteq \mathbb{F}[\bar{x}]$ be any ideal. Then, the polynomials $g_1, g_2, \cdots, g_t$ form a *Gröbner basis* for J if $J = \langle g_1, g_2, \cdots, g_s \rangle$ and $\langle LT(g_1), \cdots, LT(g_t) \rangle = \langle LT(J) \rangle$.

Lemma 1. *Let $G = \{f_1, f_2, \cdots, f_s\}$ be a Gröbner basis for an ideal $J \subseteq \mathbb{F}[\bar{x}]$ and $f \in \mathbb{F}[\bar{x}]$. Then there is a* unique *polynomial $r \in \mathbb{F}[\bar{x}]$ such that f can be written as, $f = a_1 f_1 + a_2 f_2 + \cdots + a_s f_s + r$, for $a_i \in \mathbb{F}[\bar{x}]$, and either $r = 0$ or r is an $\mathbb{F}$-linear combination of monomials, none of which is divisible by any of $LT(f_1), LT(f_2), \cdots, LT(f_s)$.*

By Lemma 1, if ideal J is given by a Gröbner basis $\{f_1, f_2, \cdots, f_s\}$ we can indeed test if $f \in J$ by computing Divide$(f; \bar{f})$ and checking if the remainder is zero. The following theorem gives us a sufficient condition for Gröbner bases.

Theorem 3 (Theorem 3, proposition 4, pp.101). [Cox92] *Let I be a polynomial ideal given by a basis $G = \{g_1, g_2, \cdots, g_s\}$ such that all pairs $i \neq j$ $LM(g_i)$ and $LM(g_j)$ are relatively prime. Then G is a Gröbner basis for I.*

An immediate consequence of the next lemma is that we can test in deterministic polynomial time if an explicitly given polynomial $f \in \mathbb{F}[\bar{x}]$ is in a monomial ideal I.

Lemma 2 (Lemma 2, Lemma 3, pp.67-68). [Cox92] *Let $I = \langle m_1, m_2, \cdots, m_s \rangle$ be a monomial ideal and $f \in \mathbb{F}[\bar{x}]$. Then, $f \in I$ if and only if each monomial of f is in I. Furthermore, a monomial m is in the ideal I if and only if there exist $i \in [s]$, such that m_i divides m.*

3 Monomial Ideal Membership

In this section we consider monomial ideal membership when f is given by an arithmetic circuit. We show that the problem is in randomized polynomial time if number of generators k for the monomial ideal I is a constant. When k is not a constant we show that it is coNP-hard and is contained in coAM$^{\mathrm{PP}}$.

Lemma 3. *Let, $I = \langle m_1, m_2, \cdots, m_k \rangle$ be a monomial ideal in $\mathbb{F}[x_1, x_2, \cdots, x_n]$. For $i \in [k]$, let $m_i = x_1^{e_{i1}} x_2^{e_{i2}} \cdots x_n^{e_{in}}$. Let $\bar{v}$ be a k-tuple given by $\bar{v} = (j_1, j_2, \cdots, j_k)$, where $j_i \in [n]$. Define the ideal, $I_{\bar{v}} = \langle x_{j_1}^{e_{1j_1}}, \cdots, x_{j_k}^{e_{kj_k}} \rangle$. Then $f \in I$ if and only if, $\forall \bar{v} \in [n]^k$, $f \in I_{\bar{v}}$.*

Proof. Given $f \in I$, it can be written as $f = p_1 m_1 + u_2 m_2 + \cdots + p_k m_k$, where $p_i \in \mathbb{F}[\bar{x}]$ for all i. Then, clearly $\forall \bar{v} \in [n]^k$, $f \in I_{\bar{v}}$. Conversely, suppose $f \notin I$. Write $f = c_1 M_1 + c_2 M_2 + \cdots + c_t M_t$, where M_i are the monomials of f and $c_i \in \mathbb{F}$. As $f \notin I$, we have $M_j \notin I$ for some j. Thus, m_i does not divide M_j for any i. Consequently, each m_i contains some x_{ℓ_i}, such that the exponent of x_{ℓ_i} is greater than the exponent of x_{ℓ_i} in M_j. Let $\{\ell_1, \ell_2, \cdots, \ell_k\}$ be k such indices. Consider the ideal $I_{\bar{w}}$, where $\bar{w} = (\ell_1, \ell_2, \cdots, \ell_k)$. By Lemma 2, $M_j \notin I_{\bar{w}}$ and hence $f \notin I_{\bar{w}}$. ■

Using Lemma 3, we generalize the Schwartz-Zippel Lemma to a form tailored for Monomial Ideal Membership.

Lemma 4. *Let $f \in \mathbb{F}[x_1, x_2, \cdots, x_n]$ be a polynomial of total degree d and $I = \langle x_1^{e_1}, x_2^{e_2}, \cdots, x_k^{e_k} \rangle$ be a monomial ideal as described in lemma 3. Fix a finite subset $S \subseteq \mathbb{F}$, and let $r_1, r_2, \cdots, r_{n-k}$ be chosen independently and uniformly at random from S. Then,* $\text{Prob}_{r_i \in S}[f(x_1, x_2, \cdots, x_k, r_1, r_2, \cdots, r_{n-k}) \in I \mid f \notin I] \leq \frac{d}{|S|}$.

Proof. Write $f = \sum_{\bar{v}} x_1^{j_1} \cdots x_k^{j_k} f_{\bar{v}}(x_{k+1}, \cdots, x_n)$, where $\bar{v} = (j_1, \cdots, j_k)$. Any term in the above expression with $j_i \geq e_i$ is already in I. Thus, it suffices to consider the sum $\hat{f}$ of the remaining terms. More precisely, Let $\mathcal{A} = [e_1 - 1] \times [e_2 - 1] \times \cdots \times [e_k - 1]$. We can write $\hat{f} = \sum_{\bar{v} \in \mathcal{A}} x_1^{j_1} \cdots x_k^{j_k} f_{\bar{v}}(x_{k+1}, \cdots, x_n)$ where $\bar{v} = (j_1, j_2, \cdots, j_k) \in \mathcal{A}$. As $\hat{f} \notin I$, not all $f_{\bar{v}}$ are identically zero. Choose and fix one such $\bar{u}$. By the Schwartz-Zippel lemma [MR01], $\text{Prob}_{r_i \in S}[f_{\bar{u}}(r_1, r_2, \cdots, r_{n-k}) = 0 \mid f_{\bar{u}}(x_{k+1}, x_{k+2}, \cdots, x_n) \not\equiv 0] \leq \frac{d}{|S|}$. For any $\bar{v} = (j_1, j_2, \cdots, j_k) \in \mathcal{A}$, notice that the monomial $x_1^{j_1} \cdots x_k^{j_k}$ is not in I. Thus, $f(x_1, x_2, \cdots, x_k, r_1, r_2, \cdots, r_{n-k}) \in I$ iff $\forall \bar{v}$, $f_{\bar{v}}(r_1, r_2, \cdots, r_{n-k}) = 0$. But $f_{\bar{u}}(r_1, r_2, \cdots, r_{n-k}) = 0$ with probability at most $d/|S|$. ■

Theorem 4. *Let $f \in \mathbb{F}[\bar{x}]$ be given by an arithmetic circuit C and the ideal $I = \langle m_1, m_2, \cdots, m_k \rangle$ generated by monomials m_i's where k is a constant. For such instances Monomial Ideal Membership can be solved in randomized polynomial time (in $n^{O(k)}$ time).*

Proof. First, we construct all the ideals, $\{I_{\bar{v}} \mid \bar{v} \in [n]^k\}$ as described in Lemma 3. Then for each such $I_{\bar{v}}$, we check if $f \in I_{\bar{v}}$. The correctness of the algorithm follows from Lemma 3. Let $I_{\bar{v}} = \langle x_1^{e_1}, x_2^{e_2}, \cdots, x_k^{e_k} \rangle$. To check $f \in I_{\bar{v}}$, we assign random values to $x_{k+1}, \cdots, x_n$ from S and then evaluate the circuit C in the ring $R = \mathbb{F}[x_1, x_2, \cdots, x_k]/I_{\bar{v}}$. To evaluate the circuit in R, we need to compute each gate operation modulo $I_{\bar{v}}$, starting from the input gates. Notice that, as $\langle x_1^{e_1}, x_2^{e_2} \cdots, x_k^{e_k} \rangle$ is a Gröbner basis for $I_{\bar{v}}$, by Lemma 1 the actual order in which we evaluate the gates is not important. Let, $e = \sum_{i=1}^{k} e_i$. Then it is easy to see that the running time of the algorithm is poly(n, s, e^k) (notice that e_i's are in unary). Furthermore, by Lemma 4,

the success probability of the algorithm is seen to be $\geq 1 - (d/|S|)$. So, it is enough to consider sampling from a set S s.t, $|S| = 2d$ using $O(\log d)$ random bits. ■

When the monomial ideal I is not generated by a constant number of monomials the monomial ideal membership problem is coNP hard over any field. The easy proof is omitted.

Theorem 5. *Given a polynomial f as an arithmetic circuit, and a monomial ideal $I = \langle m_1, m_2, \cdots, m_k \rangle$, it is* coNP*-hard to test whether $f \in I$.*

On the other hand, using standard counting arguments we can show the following upper bounds for Monomial Ideal Membership when the number of monomial generators is not restricted to a constant.

Theorem 6. *1. For $\mathbb{F} = \mathbb{Q}$, Monomial Ideal Membership is in* coAM^{PP} *where the input monomial ideal $I = \langle m_1, m_2, \cdots, m_k \rangle$ is given by a list of monomials and $f \in \mathbb{F}[\bar{x}]$ is given by an arithmetic circuit C.*

2. For $\mathbb{F} = \mathbb{F}_p$, Monomial Ideal Membership is in $\text{coNP}^{\text{Mod}_p\text{P}}$.

We now show that if the polynomial f is accessed only via a *black-box* and if f has degree*polynomial in the input size* we can still solve monomial ideal membership in randomized polynomial time (assuming I is generated by constant number of monomials). In [OT88], Ben-Or and Tiwari gave an interpolation algorithm for sparse multivariate polynomials over integers.

Theorem 7. [OT88] *Let $f \in \mathbb{Z}[x_1, x_2, \cdots, x_n]$ be a t-sparse multivariate polynomial given as a black-box (by t-sparse we mean the number of monomials in f is bounded by t), d be the degree of f, and b be a bound on the size of its coefficients. There is a deterministic algorithm that queries the black-box for values of f on different inputs and reconstructs the entire polynomial f in time* $\text{poly}(t, n, d, b)$.

Ben-Or and Tiwari's result directly gives a deterministic polynomial time algorithm for Monomial Ideal Membership when f is a t-sparse black-box polynomial over $\mathbb{Z}$, and I is any monomial ideal. The algorithm simply reconstructs f and checks if each of its monomials is in I. Next, suppose f is a black-box polynomial of small degree and I is a monomial ideal generated by constant number of monomials. The following result is an easy consequence of Lemma 3 and Theorem 7.

Theorem 8. *Let $f \in \mathbb{Z}[\bar{x}]$ of degree d given as a black-box such that b is a bound on the size of its coefficients. Suppose $I = \langle m_1, m_2, \cdots, m_k \rangle$ for constant k. Then we can test if $f \in I$ in randomized time* $\text{poly}(n^k, d^k, b)$.

4 Monomial Ideal Membership for $\Sigma\Pi\Sigma$ Circuits

Consider instances (f, I) of Monomial Ideal Membership where f is given by a $\Sigma\Pi\Sigma$ circuit with top gate of bounded fanin and $I = \langle m_1, m_2, \cdots, m_k \rangle$ a monomial ideal for constant k. By Lemma 3 this problem reduces to testing if f is in a monomial ideal

of the form $I = \langle x_1^{e_1}, x_2^{e_2}, \cdots, x_k^{e_k} \rangle$. As the quotient ring $\mathbb{F}[x_1, x_2, \cdots, x_k]/I$ is a local ring and $f \in I$ if and only if $f \equiv 0$ over the local ring $\mathbb{F}[x_1, x_2, \cdots, x_k]/I$ we can apply the Kayal-Saxena deterministic identity test [KS07] for such $\Sigma\Pi\Sigma$ circuit over local rings[2] to check this in overall time polynomial in the circuit size.

However, we develop the algorithm and its correctness proof based on Gröbner basis theory. The algorithm is essentially from [KS07]. But the Gröbner basis approach is somewhat simpler and direct. It avoids invoking properties such as chinese remaindering in local rings and Hensel lifting. The added bonus is that we get a different correctness proof for the Kayal-Saxena identity test.

Definition 2. *A $\Sigma\Pi\Sigma$ circuit C with n inputs over a field $\mathbb{F}$ computes a polynomial of the form: $C(x_1, x_2, \cdots, x_n) = \sum_{i=1}^{k} \prod_{j=1}^{d_i} L_{ij}(x_1, x_2, \cdots, x_n)$, where k is the fanin of the top Σ gate, and d_i are the fanins of the k different Π gates and L_{ij}'s are linear forms over $\mathbb{F}[x_1, x_2, \cdots, x_n]$.*

First, we transform the circuit C into another circuit C' as follows: Let $L_{ij} = \sum_{t=1}^{n} \alpha_{ijt} x_t + \beta$ for $\alpha_{ijt}, \beta \in \mathbb{F}$. We replace each such L_{ij} by $L'_{ij} = \sum_{t=1}^{n} \alpha_{ijt} x_t + \beta y$, where y is a new indeterminate. Let d be the maximum of the fanins of the Π gates. For a Π gate of fanin d_i introduce $d - d_i$ new input fanin wires each carrying y. Notice that after this transformation the resulting ideal is not a monomial ideal.

Proposition 1. *For $I = \langle x_1^{e_1}, x_2^{e_2}, \cdots, x_k^{e_k} \rangle$ and a $\Sigma\Pi\Sigma$ circuit C defined as above, $C \in I$ if and only if $C' \in \langle x_1^{e_1}, x_2^{e_2}, \cdots, x_k^{e_k}, y - 1 \rangle$.*

We can assume that in the circuit C itself every L_{ij} is of the form $\sum_{t=1}^{n} \alpha_t x_t$ and the degree of the polynomial computed at each Π gate is d. We can naturally associate to L_{ij} its coefficient vector $(\alpha_1, \alpha_2, \cdots, \alpha_n) \in \mathbb{F}^n$. A collection of linear forms is *independent* if their coefficient vectors forms a linearly independent set in $\mathbb{F}^n$.

First we fix some notation. Let R denote the polynomial ring $\mathbb{F}[x_1, x_2, \cdots, x_k]$, where k will be clear from the context where R is used. For $\alpha = (e_{k+1}, e_{k+2}, \cdots, e_n) \in \mathbb{N}^{n-k}$, let $\bar{x}^{\bar{\alpha}}$ denote $x_{k+1}^{e_{k+1}} x_{k+2}^{e_{k+2}} \cdots x_n^{e_n}$. The only monomial ordering we use is the lex-ordering defined in Definition 1 w.r.t. the order $x_1 < x_2 < \cdots < x_n$. We can consider an $f \in \mathbb{F}[x_1, \cdots, x_n]$ as a polynomial in $R[x_{k+1}, x_{k+2}, \cdots, x_n]$. More precisely, we can write $f = \sum_{\bar{\alpha} \in \mathbb{N}^{n-k}} A_{\bar{\alpha}} \bar{x}^{\bar{\alpha}}$, where $A_{\bar{\alpha}} \in \mathbb{F}[x_1, x_2, \cdots, x_k] \setminus \{0\}$. Let $\bar{\alpha_1}$ be such that $\bar{x}^{\bar{\alpha_1}}$ is the lex-largest term such that $A_{\bar{\alpha_1}} \neq 0$. Then we denote the R-leading term $A_{\bar{\alpha_1}} \bar{x}^{\bar{\alpha_1}}$ of f by $LT_R(f)$. Likewise, $LM_R(f) = \bar{x}^{\bar{\alpha_1}}$ and $LC_R(f) = A_{\bar{\alpha_1}}$ is the R-leading monomial and R-leading coefficient of f. For any $f, g \in \mathbb{F}[x_1, \cdots, x_n]$, it is clear that $LM_R(fg) = LM_R(f) LM_R(g)$, $LC_R(fg) = LC_R(f) LC_R(g)$. Let $f \in \mathbb{F}[x_1, \cdots, x_n]$ and $I = \langle f_1, f_2, \cdots, f_\ell \rangle$ be an ideal such that each f_i is in $\mathbb{F}[x_1, x_2, \cdots, x_k]$. Then the following easy lemma states a necessary and sufficient condition for f to be in I.

Lemma 5. *Let $I \subseteq \mathbb{F}[\bar{x}]$ be an ideal generated by the polynomials $f_1, f_2, \cdots, f_\ell$ such that for all $i \in [\ell]$, $f_i \in \mathbb{F}[x_1, x_2, \cdots, x_k]$. Let g be any polynomial in $\mathbb{F}[\bar{x}]$. Write $g = \sum_{\bar{\alpha} \in \mathbb{N}^{n-k}} A_{\bar{\alpha}} \bar{x}^{\bar{\alpha}}$. Then $g \in I$ if and only if for all $\bar{\alpha}$, $A_{\bar{\alpha}} \in I$.*

[2] More precisely, over local rings that allow polynomial-time arithmetic in them.

Consider polynomials $f, g \in \mathbb{F}[x_1, x_2, \cdots, x_n]$ and an ideal I such that $g \in \langle I, f\rangle$. The following useful lemma gives a sufficient condition on f under which the remainder r obtained when we invoke Divide$(g; f)$ (of Theorem 1) is in the ideal I.

Lemma 6. *Let $I = \langle f_1, f_2, \cdots, f_\ell\rangle$ be an ideal in $\mathbb{F}[x_1, \cdots, x_n]$ where $f_i \in \mathbb{F}[x_1, \cdots, x_k] = R$. Suppose f is a polynomial such that $LM(f)$ contains only variables from $\{x_{k+1}, x_{k+2}, \cdots, x_n\}$ (i.e. $LM(f) = LM_R(f)$). Then for any polynomial g in the ideal $\langle I, f\rangle$ we can write $g = qf + r$ for polynomials q and r such that $r \in I$ and no monomial of r is divisible by $LM(f)$.*

Proof. The lemma is an easy consequence of the properties of the Divide algorithm explained in Theorem 1. Notice that Divide$(g; f)$ will stop with a remainder polynomial r such that $g = qf + r$ with the property that no monomial of r is divisible by $LM(f)$. However, we only know that $r \in \langle I, f\rangle$, because both g and qf are in $\langle I, f\rangle$. We now show that r must be in I. First, as $r \in \langle I, f\rangle$ we can write $r = \sum_{i=1}^{\ell} a_i f_i + af$, for polynomials a_i and a. Following Lemma 5, we write $a_i = \sum_{\bar{\alpha}} a_{i\bar{\alpha}} \bar{x}^{\bar{\alpha}}$ for each i and also $a = \sum_{\bar{\alpha}} a_{\bar{\alpha}} \bar{x}^{\bar{\alpha}}$. Notice that we can assume $a_{\bar{\alpha}} \notin I$ for all nonzero $a_{\bar{\alpha}}$. Otherwise, we can move that term to the $\sum a_i f_i$ part. Since $LM(f)$ does not divide any monomial of r, it follows that $LM(af)$ does not occur in a nonzero term of r. Therefore, $LT(af)$ must be cancelled by some term of $\sum_{i=1}^{\ell} a_i f_i$. Clearly, $LT(af)$ is of the form $c \cdot a_{\bar{\beta}} \bar{x}^{\bar{\alpha}}$ for some α, β, where $LC(f) = c \in \mathbb{F}$ and $a_{\bar{\beta}} = LC_R(a)$. Now, in $\sum_{i=1}^{\ell} a_i f_i$ the coefficient of $\bar{x}^{\bar{\alpha}}$ is $\sum_{i=1}^{\ell} a_{i\bar{\alpha}} f_i$ which must be equal to $-c \cdot a_{\bar{\beta}}$. Since $c \in \mathbb{F}$ it follows that $a_{\bar{\beta}}$ is in I contradicting the assumption that none of the nonzero $a_{\bar{\gamma}}$ is in I. ∎

Again, let $I = \langle f_1, f_2, \cdots, f_\ell\rangle$ such that the f_i are in $\mathbb{F}[x_1, x_2, \cdots, x_k]$. Consider two polynomials f and g such that $LM(f)$ contains only variables from $x_{k+1}, x_{k+2}, \cdots, x_n$ and either $LM(f) > LM(g)$ or $LM_R(f) = LM_R(g)$ and $LC_R(g) \in I$. Then, g is in the ideal $\langle I, f\rangle$ if and only if $g \in I$.

Lemma 7. *Let $I = \langle f_1, f_2, \cdots, f_\ell\rangle$ be an ideal in $\mathbb{F}[x_1, \cdots, x_n]$ such that each f_i is in $\mathbb{F}[x_1, x_2, \cdots, x_k] = R$. Suppose f is a polynomial such that $LM(f)$ is over the variables only from $\{x_{k+1}, x_{k+2}, \cdots, x_n\}$ (i.e. $LM(f) = LM_R(f)$). Then for any polynomial g such that either $LM(f) > LM(g)$, or $LM_R(f) = LM_R(g)$ and $LC_R(g) \in I$, g is in the ideal $\langle I, f\rangle$ if and only if g is in the ideal I.*

Proof. Suppose $g \in \langle I, f\rangle$ and $g \notin I$. We can write $g = a + bf$, for polynomials a and b, where $a \in I$. Also, we can assume that $b \notin I$, for otherwise $g \in I$ and we are done. Let $b = \sum_{\bar{\alpha} \in \mathbb{N}^{n-k}} b_{\bar{\alpha}} \bar{x}^{\bar{\alpha}}$, where $b_{\bar{\alpha}} \in \mathbb{F}[x_1, x_2, \cdots, x_k]$ and we can assume $b_{\bar{\alpha}} \notin I$ for all $\bar{\alpha}$ (otherwise we can move that term as part of a). Notice that $LT_R(bf) = LT_R(b) \cdot LT_R(f) = cb_{\bar{\beta}} LM_R(b) LM_R(f) = cb_{\bar{\beta}} \bar{x}^{\bar{\gamma}}$ for some $\bar{\gamma}$ and for some $b_{\bar{\beta}}$, where $c = LC_R(f) \in \mathbb{F}$. Since $b_{\bar{\beta}} \notin I$ it follows that $LC_R(bf) \notin I$. Write $a = \sum_{\bar{\alpha} \in \mathbb{N}^{n-k}} a_{\bar{\alpha}} \bar{x}^{\bar{\alpha}}$. By Lemma 5, $a \in I$ implies each $a_{\bar{\alpha}} \in I$. In particular, $a_{\bar{\gamma}} \in I$ and is *not* equal to $-LC_R(b \cdot f) = -cb_{\bar{\beta}}$ as $b_{\bar{\beta}} \notin I$. Thus, the monomial $LM_R(bf)$ survives in $a + bf$. It follows that $LM_R(g) = LM_R(a + bf) \geq LM_R(bf) \geq LM_R(f)$ which forces $LM_R(f) = LM_R(g)$ and $LC_R(g) \in I$ by assumption. If $b \notin R$ then $LM_R(b \cdot f) > LM_R(f)$ which implies $LM_R(g) > LM_R(f)$ contradicting assumption.

If $b \in R$ then $LT_R(g) = LT_R(a + bf) = (a_{\bar{\alpha}} + b)LM_R(f)$ for some $a_{\bar{\alpha}}$, which forces $b \in I$ because both $LT_R(g), a_{\bar{\alpha}} \in I$. ∎

Let $I \subseteq \mathbb{F}[x_1, \cdots, x_n]$ be an ideal and g_1, g_2 are two polynomials such that f is in the ideals $\langle I, g_1\rangle$ and $\langle I, g_2\rangle$. Using some Gröbner basis theory we give a sufficient condition on I, g_1 and g_2 under which we can infer that f is in the ideal $\langle I, g_1g_2\rangle$.

Lemma 8. *Let $I = \langle f_1, f_2, \cdots, f_\ell\rangle$ be an ideal of $\mathbb{F}[x_1, x_2, \cdots, x_n]$, where f_i are polynomials in $\mathbb{F}[x_1, x_2, \cdots, x_k]$. Suppose g_1 and g_2 are polynomials such that: $g_2 = \prod_{i=1}^{d_2}(x_{k+1} - \alpha_i)$, where each α_i is a linear form over $x_1, x_2, \cdots, x_k$, and the leading term $LT(g_1)$ of g_1 has only variables from $\{x_{k+2}, x_{k+3}, \cdots, x_n\}$. Then $f \in \langle I, g_1g_2\rangle$ if and only if $f \in \langle I, g_1\rangle$ and $f \in \langle I, g_2\rangle$.*

Proof. The forward implication is obvious. We prove the reverse direction. Suppose $f \in \langle I, g_1\rangle$ and $f \in \langle I, g_2\rangle$. As $f \in \langle I, g_2\rangle$, we can write $f = a + bg_2$, where $a \in I$ and b is an arbitrary polynomial. Notice that it suffices to prove bg_2 is in the ideal $\langle I, g_1g_2\rangle$. Now, since $f \in \langle I, g_1\rangle$ and $a \in I$ it follows that $bg_2 = f - a \in \langle I, g_1\rangle$. By applying Lemma 6 to ideal I and polynomial g_1 observe that we can write $bg_2 = \alpha g_1 + \beta$, where β is a polynomial in I such that none of the monomials of β is divisible by $LT(g_1)$. We have the following equation $b \cdot \prod_{j=1}^{d_2}(x_{k+1} - \alpha_j) = \alpha g_1 + \beta$. Substituting $x_{k+1} = \alpha_1$ in the above equation, we get $(\alpha g_1)|_{x_{k+1}=\alpha_1} = -\beta|_{x_{k+1}=\alpha_1}$. Notice that $LT(g_1|_{x_{k+1}=\alpha_1}) = LT(g_1)$, as $LT(g_1)$ contains variables only from $x_{k+2}, \cdots, x_n$. Thus the above substitution implies $LT(\beta|_{x_{k+1}=\alpha_1}) = -LT((\alpha g_1)|_{x_{k+1}=\alpha_1}) = -LT(\alpha|_{x_{k+1}=\alpha_1}) \cdot LT(g_1|_{x_{k+1}=\alpha_1}) = -LT(\alpha|_{x_{k+1}=\alpha_1}) \cdot LT(g_1)$.

Thus $LM(g_1)$ divides $LM(\beta|_{x_{k+1}=\alpha_1})$. On the other hand, since $LM(g_1)$ does not divide any monomial of β, $LM(g_1)$ cannot divide any monomial of $LM(\beta|_{x_{k+1}=\alpha_1})$ as the substitution only introduces variables from $\{x_1, \cdots, x_k\}$. This gives a contradiction unless $\beta|_{x_{k+1}=\alpha_1} = 0$, which in turn implies $\alpha|_{x_{k+1}=\alpha_1} = 0$. Thus we have proved that $(x_{k+1} - \alpha_1)$ is a factor of both α and β. This leads us to the following similar identity: $b \cdot \prod_{j=2}^{d_2}(x_{k+1} - \alpha_j) = \alpha_1 g_1 + \beta_1$, where $\alpha_1 = \alpha/(x_{k+1} - \alpha_1)$ and $\beta_1 = \beta/(x_{k+1} - \alpha_1)$. Clearly, by repeating the above argument we finally get, $b = \alpha' g_1 + \beta'$, for some polynomials α' and β' where $\alpha = \alpha' g_2$ and $\beta = \beta' g_2$. Putting it together we get $bg_2 = \alpha' g_1 g_2 + \beta' g_2 = \alpha' g_1 g_2 + \beta$. As $\beta \in I$, it follows that bg_2 is in the ideal $\langle I, g_1g_2\rangle$. ∎

Let $I = \langle P_1, P_2, \cdots, P_k\rangle$ be an ideal in $\mathbb{F}[x_1, \cdots, x_n]$ such that $P_i \in \mathbb{F}[x_1, x_2, \cdots, x_i]$ and $LT(P_i) = x_i^{d_i}$ for each i. For $i \neq j$ the leading terms $LT(P_i) = x_i^{d_i}$ and $LT(P_j) = x_j^{d_j}$ are clearly relatively prime. Therefore by Theorem 3, it follows that $\{P_1, P_2, \cdots, P_k\}$ is in fact a Gröbner basis for I. We summarize this observation.

Lemma 9. *Let $I = \langle P_1, P_2, \cdots, P_k\rangle$ be an ideal in $\mathbb{F}[x_1, \cdots, x_n]$ such that each P_i is in $\mathbb{F}[x_1, x_2, \cdots, x_i]$ and $LT(P_i) = x_i^{d_i}$. Then $\{P_i\}_{i\in[k]}$ is a Gröbner basis for I.*

Let $f \in \mathbb{F}[x_1, x_2, \cdots, x_k]$ and d be the maximum of $\deg(f)$ and $\deg(P_i), 1 \leq i \leq k$. We can invoke Divide$(f; P_1, P_2 \cdots, P_k)$ (Theorem 1) to test whether $f \in I$. By Fact 2 the running time for this test is $O(d^k)$. Now we state the main theorem of this section.

Theorem 9. *Let $C \in \mathbb{F}[x_1, x_2 \cdots, x_n]$ be given by a $\Sigma\Pi\Sigma(\ell, d)$ circuit for a constant ℓ and $I = \langle m_1, m_2, \cdots, m_k \rangle$ be a monomial ideal for constant k. For such instances, Monomial Ideal Membership can be checked in deterministic polynomial time. Specifically, the running time is bounded by $n^k \text{poly}(n, d^{max\{\ell,k\}})$.*

By Lemma 3 it clearly suffices to give a polynomial-time deterministic algorithm for testing if a $\Sigma\Pi\Sigma(\ell, d)$ circuit C is in a monomial ideal of the form $\langle x_1^{e_1}, \cdots, x_k^{e_k} \rangle$. As explained before, using a new indeterminate y we can transform C to C' with homogeneous linear forms, and $C \in I$ if and only if $C' \in \langle x_1^{e_1}, \cdots, x_k^{e_k}, y - 1 \rangle$. The following theorem along with Lemma 3 yields Theorem 9.

Theorem 10. *Let C be a given $\Sigma\Pi\Sigma(\ell, d)$ circuit for a constant ℓ and $I = \langle P_1, P_2, \cdots, P_k \rangle$ be an ideal in $\mathbb{F}[x_1, \cdots, x_n]$ such that $P_i \in \mathbb{F}[x_1, x_2, \cdots, x_i]$ and $LT(P_i) = x_i^{d_i}$ for each i. Further, suppose $d_i \leq d$ for all $i \in [k]$. Then testing if $C \in I$ can be done in deterministic time* $\text{poly}(d^{max\{\ell,k\}})$.

Proof. We can assume that all linear forms appearing in C and C itself is a homogeneous degree d polynomial. By Lemma 9, the generating set for I is a Gröbner basis. Let $C(x_1, x_2, \cdots, x_n) = \sum_{i=1}^{\ell} T_i$. For all $i \in [\ell]$, $T_i = \prod_{j=1}^{d} L_{ij}$, where L_{ij}'s are the linear forms over $\mathbb{F}[x_1, x_2, \cdots, x_n]$.

If $\ell = 1$, then $C = T_1$. Let $g(x_1, x_2, \cdots, x_k)$ be the product of those linear forms of T_1 using only variables from $\{x_1, x_2, \cdots, x_k\}$. Clearly, $g(x_1, x_2, \cdots, x_k)$ has at most d^k monomials. We explicitly compute g by multiplying out all such linear forms. By Lemma 5, clearly $C \in I$ if and only if $g \in I$, which can be checked in time $\text{poly}(d^k)$ following the Fact 2. Now, assume $\ell > 1$. If all the linear forms appearing in $T_1, T_2, \cdots, T_\ell$ are only over $\{x_1, x_2, \cdots, x_k\}$, then again the ideal membership testing is easy. Because, in time $\text{poly}(d^k)$ we can write C itself as an $\mathbb{F}$-linear combination of monomials in $x_1, x_2, \cdots, x_k$ and apply Fact 2 to check if $f \in I$ in time $\text{poly}(d^k)$.

Now we consider the general case. By inspection we can write each $T_i = \beta_i T_i'$ where the β_i are products of linear forms over only $x_1, x_2, \cdots, x_k$, whereas each linear form in T_i' involves at least one other variable.[3] If $\beta_i \in I$ (which we can test in polynomial time using Fact 2) we drop the term T_i from the sum $\sum_{i=1}^{\ell} T_i$. This enables us to write C as $C = \beta_1 T_1' + \beta_2 T_2' + \cdots + \beta_m T_m'$ for some $m \leq \ell$, where we have assumed for simplicity of notation that $\beta_i \notin I$ for first m terms.

As before, let $R = \mathbb{F}[x_1, x_2, \cdots, x_k]$. W.l.o.g, assume that $LM_R(T_1') \geq LM_R(T_i')$ for all $i \in [2, 3, \cdots, m]$. We can determine $LT_R(T_i')$ for each T_i' in polynomial time since they are given as product of linear forms. Thus, $LM_R(T_1') \geq LM_R(C)$. Now, let $r \in R$ be the coefficient of $LM_R(T_1')$ in C. We can compute r in polynomial time by computing the coefficient of $LM_R(T_1')$ in each T_i' and adding them up. Then we check that $r \in I$ (which is a necessary condition for C to be in I by Lemma 5). By Fact 2 we can check $r \in I$ in time $\text{poly}(d^k)$. It is clear that, either $LM_R(T_1') > LM_R(C)$ or $LM_R(T_1') = LM_R(C)$ and $r \in I$. Thus, by the Lemma 7, $C \in I$ if and only if $C \in \langle I, T_1' \rangle$. Next, we group the linear forms in T_1': let, $T_1' = T_{11} T_{12} \cdots T_{1t}$, such that for all $i \in [t]$, $T_{1i} = \prod_{j=1}^{i} (L_i + m_{is_j})$, where $\{L_i\}_{i=1}^{t}$ are *distinct linear forms* in

[3] If there are no linear forms contributing to the product β_i (respectively, T_i') we will set it to 1.

$\mathbb{F}[x_{k+1}, \cdots, x_n]$ and m_{is_j}'s are linear forms in $\mathbb{F}[x_1, \cdots, x_k]$. Notice that the polynomials T_{1i} are relatively prime to each other.

Then we compute t linear transformations $\{\sigma_1, \sigma_2, \cdots, \sigma_t\}$ from $\mathbb{F}^n$ to $\mathbb{F}^n$ with the following property: for $i \in [t]$, σ_i fixes $\{x_i\}_{i=1}^k$, maps L_i to x_{k+1} and maps $\{x_{k+2}, x_{k+3}, \cdots, x_n\}$ to some suitable linear forms in such a way that, σ_i is an invertible linear transformation. As L_i's are over $\{x_{k+1}, \cdots, x_n\}$, it is easy to see that such σ_i exist and are easy to compute. Let $C_1 = \sum_{j \in [\ell] \setminus \{1\}} T_j$. For $i \in [t]$, let $C_{1i} = \sigma_i(C_1)$ and let I_{1i} be the ideal $\langle I, \sigma_i(T_{1i}) \rangle$. The algorithm recursively checks that each of the $\Sigma\Pi\Sigma(\ell - 1, d)$ circuits C_{1i} is in the ideal I_{1i}, and declares $C \in I$ if and only if $C_{1i} \in I_{1i}$ for each i. Notice that the ideal I_{1i} has generating set $G = \{P_1, P_2, \cdots, P_k, P_{k+1}\}$, where $P_{k+1} \in \mathbb{F}[x_1, x_2, \cdots, x_{k+1}]$ and $LM(P_{k+1}) = x_{k+1}^{d_{k+1}}$. By Lemma 9, G is a Gröbner basis for I_{1i}.

To argue correctness of the algorithm, we claim that for each $s : 1 \leq s \leq t$, we have $C \in \langle I, T_{11}T_{12} \cdots T_{1s} \rangle$ if and only if $C_{1i} \in I_{1i}$ for $1 \leq i \leq s$. In particular, $C \in \langle I, T_1' \rangle$ if and only if $C_{1i} \in I_{1i}$ for $1 \leq i \leq t$. We now establish the claim. If $C \in \langle I, T_{11}T_{12} \cdots T_{1s} \rangle$ then clearly $C \in \langle I, T_{1i} \rangle$ for each $1 \leq i \leq s$. As each σ_i is an invertible linear map it follows in turn that $\sigma_i(C) \in \langle I, \sigma_i(T_{1i}) \rangle = I_{1i}$ for $1 \leq i \leq s$. Since $C_{1i} = \sigma_i(C) - \sigma_i(T_1)$ and $\sigma_i(T_1) \in \langle \sigma_i(T_{1i}) \rangle$ it follows that $C_{1i} \in I_{1i}$ for $1 \leq i \leq s$. We prove the other direction of the claim by induction on s. The base case $s = 1$ is trivial. Inductively assume it is true for $s-1$. I.e. if $C_{1i} \in I_{1i}$ for $1 \leq i \leq s-1$ then $C \in \langle I, T_{11}T_{12} \cdots T_{1(s-1)} \rangle$. We now prove the induction step for s. Suppose $C_{1i} \in I_{1i}$ for $1 \leq i \leq s$. Let $T = T_{11}T_{12} \cdots T_{1(s-1)}$. By induction hypothesis we have $C \in \langle I, T \rangle$. Furthermore, $C_{1s} \in I_{1s}$ implies by definition that $C \in \langle I, T_{1s} \rangle$. Now we apply the linear map σ_s to obtain $\sigma_s(C) \in \langle I, \sigma_s(T) \rangle$ and $\sigma_s(C) \in \langle I, \sigma_s(T_{1s}) \rangle$. The map σ_s ensures that $LT(T_{1s})$ is of the form $x_{k+1}^{\deg T_{1s}}$. Furthermore, by the definition of σ_s it follows that $LT(\sigma_s(T))$ has only variables in $\{x_{k+2}, \cdots, x_n\}$. Letting $g_1 = \sigma_s(T)$ and $g_2 = \sigma_s(T_{1s})$ in Lemma 8, it follows immediately that $\sigma_s(C) \in \langle I, \sigma_s(T \cdot T_{1s}) \rangle$ which implies the induction step since σ_s is invertible.

We show that the above algorithm runs in time $\text{poly}(n, d^{\max\{\ell,k\}})$. Let $T(\ell, n)$ be the time to test $C \in I$. It is easy to see from the description of the algorithm that, $T(\ell, n) \leq tT(\ell - 1, n) + \text{poly}(n, d^k)$. Hence, $T(\ell, n) = \text{poly}(n, d^{\max\{\ell,k\}})$ as $t = O(d)$. ∎

5 Bounded Variable Ideal Membership

We now consider the ideal membership problem when $I = \langle f_1, \cdots, f_\ell \rangle$ such that $f_i \in \mathbb{F}[x_1, \cdots, x_k]$ for a constant k and the polynomial f is given by an arithmetic circuit. We call this variant *bounded variable Ideal Membership*. We first recall Hermann's fundamental result.

Theorem 11. [He26] *Consider polynomials $f, f_1, f_2, \cdots, f_m \in \mathbb{F}[x_1, x_2, \cdots, x_k]$ for a field $\mathbb{F}$ such that $\max\{\deg(f_1), \deg(f_2), \cdots, \deg(f_m), \deg(f)\} \leq d$. If f is in the ideal $I = \langle f_1, f_2, \cdots, f_m \rangle$ then f can be expressed as $f = \sum_{i=1}^m g_i f_i$ where $\deg(g_i) \leq (2d)^{2^k}$ for each i.*

Suppose f is given explicitly as an $\mathbb{F}$-linear combination of terms. Using the bounds of Hermann's theorem, Hermann's algorithm treats the coefficients of g_i as unknowns

and does membership testing in $\langle f_1, f_2, \cdots, f_m \rangle$ by solving a system of linear equations with $m(2d)^{k2^k}$ unknowns using Gaussian elimination in time $m^{O(1)}(2d)^{O(k2^k)}$. Similarly, for an explicitly given $f \in \mathbb{F}[x_1, \cdots, x_n]$, using Lemma 5 in combination with Hermann's algorithm we can test if $f \in \langle f_1, f_2, \cdots, f_m \rangle$ in time polynomial in the size of f and $m^{O(1)}(2d)^{O(k2^k)}$. If k is a constant, this gives a polynomial running time bound.

A natural question here is the complexity of Ideal Membership when f is given by an arithmetic circuit whose membership we want to test in ideal $I = \langle f_1, f_2, \cdots, f_m \rangle$, where $f_i \in \mathbb{F}[x_1, \cdots, x_k]$ for constant k. We consider polynomials f computed by arithmetic circuits of polynomial degree in the input size. We can follow essentially the same proof idea in the Theorem 4. Notice that $f \in I$ if and only if $f \equiv 0$ in the ring $R[x_{k+1}, x_{k+2}, \cdots, x_n]$ where $R = \mathbb{F}[x_1, x_2, \cdots, x_k]/I$. We need the following proposition about zeros of a univariate polynomial over an arbitrary ring.

Proposition 2. *Let R be an arbitrary commutative ring containing a field $\mathbb{F}$. If $f \in R[x]$ is a nonzero polynomial of degree d then $f(a) = 0$ for at most d distinct values of $a \in \mathbb{F}$.*

Proof. Suppose $f(a_i) = 0$ for distinct points $a_i \in \mathbb{F}, 1 \leq i \leq d+1$. Write $f(x) = (x - a_1)q(x)$ for $q(x) \in R[x]$. Dividing $q(x)$ by $x - a_2$ yields $q(x) = (x - a_2)q'(x) + q(a_2)$, for some $q'(x) \in R[x]$. Thus, $f(x) = (x - a_1)(x - a_2)q'(x) + (x - a_1)q(a_2)$. Putting $x = a_2$ yields $(a_2 - a_1)q(a_2) = 0$. It forces $q(a_2) = 0$ since $a_2 - a_1 \neq 0$ is a unit in $\mathbb{F}$. Therefore, $f(x) = (x - a_1)(x - a_2)q'(x)$. Repeating this argument for the other a_i yields $f(x) = g(x) \prod_{i=1}^{d+1}(x - a_i)$ for some nonzero polynomial $g(x) \in R[x]$, which forces $\deg(f) \geq d + 1$ contradicting the assumption. ∎

An easy induction argument yields the following analog of the Schwartz-Zippel test for arbitrary commutative rings.

Lemma 10. *Let R be an arbitrary commutative ring containing a field $\mathbb{F}$. Let $g \in R[x_1, x_2, \cdots, x_m]$ be any polynomial of degree at most d. If $g \not\equiv 0$, then for any finite subset A of $\mathbb{F}$ we have* $\text{Prob}_{a_1 \in A, \cdots, a_m \in A}[g(a_1, a_2, \cdots, a_m) = 0 \mid g \not\equiv 0] \leq \frac{md}{|A|}$.

Now we describe our ideal membership test: Choose and fix $S \subseteq F$ of size $2(n - k)d$ and randomly assign values from S to the variables in $\{x_{k+1}, \cdots, x_n\}$. Clearly f, given by a polynomial degree arithmetic circuit C, is in I if and only if $f \equiv 0$ in $R[x_{k+1}, x_{k+2}, \cdots, x_n]$, where $R = \mathbb{F}[x_1, x_2, \cdots, x_k]/I$. After the random substitution we are left with an arithmetic circuit $C'(x_1, \cdots, x_k)$. By Lemma 10 if $f \notin I$ then $C'(x_1, \cdots, x_k) \notin I$ with probability at least $1/2$. It suffices to test if C' is in I. As C' is of polynomial degree d and k is a constant, notice that C' is an $\mathbb{F}$-linear combination of at most d^k monomials. Clearly, we can test if $C' \in \langle f_1, \cdots, f_\ell \rangle$ in polynomial time using Hermann's algorithm as k is a constant. Similarly, Theorem 8 for black-box polynomials can be easily extended to bounded variable Ideal Membership.

Finally, when f is given by a $\Sigma\Pi\Sigma$ circuit with bounded fanin output gate, we can follow the algorithm in the proof of Theorem 10 to end up with the problem of testing if a polynomial g given by a $\Pi\Sigma$ circuit is in an ideal $\langle f_1, \cdots, f_\ell \rangle$, where f_i are all in $\mathbb{F}[x_1, \cdots, x_t]$ for a constant t. Again, we can apply Hermann's algorithm to check

this in time polynomial in $(m+n+d)^{O(t2^t)}$ which is a polynomial time bound as t is constant.

Theorem 12. *Let $I = \langle f_1, f_2, \cdots, f_m \rangle$ be an ideal in $\mathbb{F}[x_1, x_2, \cdots, x_n]$ where each $f_i \in \mathbb{F}[x_1, x_2, \cdots, x_k]$ for constant k. If f be a polynomial given by an arithmetic circuit of polynomial degree, then in randomized polynomial time we can test if $f \in I$. This result holds even if f is given by a black-box and the degree of f is polynomial in the input size. Further, if f is given by a $\Sigma\Pi\Sigma(\ell, d)$ circuit with ℓ constant, then we can test whether $f \in I$ in deterministic polynomial time.*

References

[AB03] Agrawal, M., Biswas, S.: Primality and identity testing via Chinese remaindering. J. ACM 50(4), 429–443 (2003)

[AKS04] Agrawal, M., Kayal, N., Saxena, N.: Primes is in P. Annals of Mathematics 160(2), 781–793 (2004)

[Cox92] Cox, D., Little, J., O'Shea, D.: Ideals, Varieties and Algorithms. Undergraduate Text in Mathematics. Springer, Heidelberg (1992)

[DS05] Dvir, Z., Shpilka, A.: Locally Decodable Codes with 2 queries and Polynomial Identity Testing for depth 3 circuits. In: Proc. of the 37th annual ACM Sym. on Theory of computing (2005)

[He26] Hermann, G.: Die Frage der endlich viel Schritte in der Theorie der Polynomideale. Math. Annalen 95, 736–788 (1926)

[KI03] Kabanets, V., Impagliazzo, R.: Derandomization of polynomial identity tests means proving circuit lower bounds. In: Proc. of the thirty-fifth annual ACM Sym. on Theory of computing, pp. 355–364 (2003)

[KS07] Kayal, N., Saxena, N.: Polynomial Identity Testing for Depth 3 Circuits. Computational Complexity 16(2), 115–138 (2007)

[LFKN92] Lund, C., Fortnow, L., Karloff, H., Nisan, N.: Algebraic methods for interactive proof systems. Journal of the ACM 39(4), 859–868 (1992)

[Mayr89] Mayr, E.: Membership in polynomial ideals over $\mathbb{Q}$ is exponential space complete. In: Cori, R., Monien, B. (eds.) STACS 1989. LNCS, vol. 349, Springer, Heidelberg (1989)

[MM82] Mayr, E., Meyer, A.: The complexity of word problem for commutative semigroups and polynomial ideals. Adv. Math. 46, 305–329 (1982)

[MR01] R. Motwani and P. Raghavan. Randomized Algorithm. Cambridge (2001)

[OT88] Ben-Or, M., Tiwari, P.: A Deterministic Algorithm For Sparse Multivariate Polynomial Interpolation. In: Proc. of the 20th annual ACM Sym. on Theory of computing, pp. 301–309 (1988)

[Sch80] Schwartz, J.T.: Fast Probabilistic algorithm for verification of polynomial identities. J. ACM 27(4), 701–717 (1980)

[Su98] Sudan, M.: Lectures on Algebra and Computation (6.966), Lecture 12,13,14 (1998)

[Zip79] Zippel, R.: Probabilistic algorithms for sparse polynomials. In: Proc. of the Int. Sym. on Symbolic and Algebraic Computation, pp. 216–226 (1979)

On the Fault Testing for Reversible Circuits

Satoshi Tayu, Shigeru Ito, and Shuichi Ueno

Department of Communications and Integrated Systems
Tokyo Institute of Technology, Tokyo 152–8550–S3–57, Japan
{tayu,ueno}@lab.ss.titech.ac.jp

Abstract. This paper shows that it is NP-hard to generate a minimum complete test set for stuck-at faults on the wires of a reversible circuit. We also show non-trivial lower bounds for the size of a minimum complete test set.

1 Introduction

Reversible circuits, which permute the set of input vectors, have potential applications in nanocomputing [3], low power design [1], digital signal processing [6], and quantum computing [4]. This paper shows that given a reversible circuit C, it is NP-hard to generate a minimum complete test set for stuck-at faults which fix the values of wires in C to either 0 or 1. This is the first result on the complexity of fault testing for reversible circuits, as far as the authors know. We also show non-trivial lower bounds for the size of a minimum complete test set.

A gate is *reversible* if the Boolean function it computes is bijective. If a reversible gate has k input and output wires, it is called a $k \times k$ *gate*. A circuit is *reversible* if all gates are reversible and are interconnected without funout or feedback. If a reversible circuit has n input and output wires, it is called an $n \times n$ *circuit*.

We shall focus our attention to detecting faults in a reversible circuit C which cause wires to be stuck-at-0 or stuck-at-1. Let $W(C)$ be the set of all wires of C. $W(C)$ consists of all output wires of C and input wires to the gates in C. $W(C)$ is the set of all possible fault locations in C. For an $n \times n$ reversible circuit C, a test is an input vector in $\{0,1\}^n$. A test set is said to be *complete* for C if it can detect all possible single and multiple stuck-at faults on $W(C)$. Patel, Hayes, and Markov [5] showed that for any reversible circuit C, there exists a complete test set for C. Let $\tau(C)$ be the minimum cardinality of a complete test set for C.

We first show that it is NP-hard to compute $\tau(C)$ for a given reversible circuit C. Let MTS (Minimum Test Size) be a problem of deciding if $\tau(C) \leq B$ for a given reversible circuit C and integer B. We show in Section 3 that MTS is NP-complete.

Patel, Hayes, and Markov [5] showed a general upper bound for $\tau(C)$ as follows. They showed that

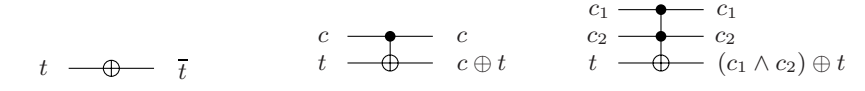
$$\tau(C) = O(\log |W(C)|) \tag{1}$$

T. Tokuyama (Ed.): ISAAC 2007, LNCS 4835, pp. 812–821, 2007.

for any reversible circuit C. We show the first non-trivial existential lower bound for $\tau(C)$. We show in Section 4 that there exists a reversible circuit C such that

$$\tau(C) = \Omega(\log\log|W(C)|). \tag{2}$$

A k-CNOT gate is a $(k+1)\times(k+1)$ reversible gate. It passes some k inputs, referred to as control bits, to the outputs unchanged, and inverts the remaining input, referred to as target bit, if the control bits are all 1. The 0-CNOT gate is just an ordinary NOT gate. A CNOT gate is a k-CNOT gate for some k. Some CNOT gates are shown in Fig. 1, where a control bit and target bit are denoted by a black dot and ring-sum, respectively. A *CNOT circuit* is a reversible circuit consisting of only CNOT gates. A *k-CNOT circuit* is a CNOT circuit consisting of only k-CNOT gates. Any Boolean function can be implemented by a CNOT circuit since the 2-CNOT gate can implement the NAND function.



(a) 0-CNOT gate. (b) 1-CNOT gate. (c) 2-CNOT gate.

Fig. 1. CNOT gates

Chakraborty [2] showed that

$$\tau(C) \le n \tag{3}$$

if C is an $n\times n$ CNOT circuit with no 0-CNOT or 1-CNOT gate. We show in Section 5 that there exists an $n\times n$ 2-CNOT circuit C such that

$$\tau(C) = \Omega(\log n). \tag{4}$$

It is an interesting open problem to close the gaps between the upper bounds (1) and (3), and our lower bounds (2) and (4), respectively.

2 Complete Test Sets

A wire w of a reversible circuit C is said to be *controllable* by a test set T if the value of w can be set to both 0 and 1 by T. A set of wires $S \subseteq W(C)$ is said to be *controllable* by T if each wire of S is controllable by T. The following characterization for a complete test set is shown in [5].

Theorem I. *A test set T for a reversible circuit C is complete if and only if $W(C)$ is controllable by T.* □

3 NP-Completeness of MTS

The purpose of this section is to prove the following:

Theorem 1. *MTS is NP-complete.*

Proof. A minimum complete test set T for a reversible circuit C can be verified in polynomial time, since $|T| = O(\log |W(C)|)$ by (1). Thus MTS is in NP.

We show a polynomial time reduction from 3SAT, a well-known NP-complete problem, to MTS. Let $\boldsymbol{x} = (x_1, , x_2, \ldots, x_n)$ and

$$\phi(\boldsymbol{x}) = \bigwedge_{j=1}^{m} \rho_j$$

be a Boolean function in conjunctive normal form in which each clause ρ_j has 3 literals for $j \in [m] = \{1, 2, \ldots, m\}$. For a Boolean variable x, literals $\overline{x}$ and x are denoted by x^0 and x^1, respectively.

We use generalized CNOT gates for simplicity. A *generalized k-CNOT gate* has k control bits $x_1, \ldots, x_k$ and a target bit t. The output of the target bit is defined as

$$(x_1^{\alpha_1} \wedge x_2^{\alpha_2} \wedge \cdots \wedge x_k^{\alpha_k}) \oplus t.$$

A control bit x_i is said to be positive if $\alpha_i = 1$, and negative if $\alpha_i = 0$. Notice that a CNOT gate is a generalized CNOT gate with no negative control bit. Notice also that a negative control bit is equivalent to a positive control bit with a 0-CNOT gate on the input and output wires. A *generalized CNOT* [*k-CNOT*] *circuit* is a reversible circuit consisting of only generalized CNOT [k-CNOT] gates.

We first construct a generalized CNOT gate G_j for each clause ρ_j. Let

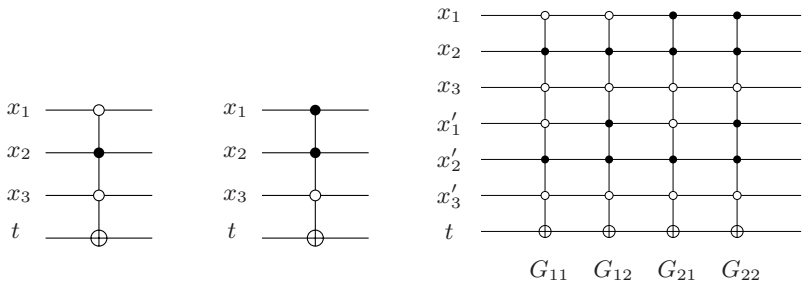
$$\rho_j = x_{j1}^{\sigma_{j1}} \vee x_{j2}^{\sigma_{j2}} \vee x_{j3}^{\sigma_{j3}},$$

where $\sigma_{jl} \in \{0, 1\}$ and $x_{jl} \in \{x_i | i \in [n]\}$ for $l \in [3]$. We construct a generalized 3-CNOT gate G_j for ρ_j as follows. The gate G_j has 3 control bits x_{j1}, x_{j2}, x_{j3}, and a target bit t. A control bit x_{jl} is defined to be positive if $\sigma_{jl} = 0$, and negative if $\sigma_{jl} = 1$. For an $n \times n$ circuit C and an input vector $\boldsymbol{v} \in \{0, 1\}^n$, we denote by $C(\boldsymbol{v})$ the output vector of C for $\boldsymbol{v}$. The following lemma is immediate from the definition of G_j.

Lemma 1. $G_j(x_{j1}, x_{j2}, x_{j3}, t) = (x_{j1}, x_{j2}, x_{j3}, \overline{\rho_j} \oplus t)$. □

Lemma 1 means that G_j changes the target bit t for input vector $(x_{j1}, x_{j2}, x_{j3}, t)$ if and only if $\rho_j(x_{j1}, x_{j2}, x_{j3}) = 0$. As an example, for a Boolean function:

$$\left.\begin{aligned} \psi(x_1, x_2, x_3) &= \rho_1 \wedge \rho_2, \\ \rho_1 &= x_1 \vee \overline{x_2} \vee x_3, \text{ and} \\ \rho_2 &= \overline{x_1} \vee \overline{x_2} \vee x_3, \end{aligned}\right\} \tag{5}$$



(a) G_1. (b) G_2. (c) 6-CNOT circuit $C(\psi)$.

Fig. 2. Generalized 3-CNOT gates and 6-CNOT circuit

generalized 3-CNOT gates G_1 and G_2 are shown in Fig. 2(a) and (b), where a negative control bit is denoted by an empty circle.

We next construct a $(2n+1) \times (2n+1)$ generalized 6-CNOT circuit $C(\phi)$ for ϕ. For $\boldsymbol{x} = (x_1, , x_2, \ldots, x_n)$, $\boldsymbol{y} = (y_1, , y_2, \ldots, y_n) \in \{0,1\}^n$, and $t \in \{0,1\}$, let $\overline{\boldsymbol{x}} = (\overline{x}_1, \overline{x}_2, \ldots, \overline{x}_n)$ and $(\boldsymbol{x}, \boldsymbol{y}, t) = (x_1, , x_2, \ldots, x_n, y_1, , y_2, \ldots, y_n, t)$. Let G'_j be a copy of G_j with control bits $x'_{j1}, x'_{j2}, x'_{j3}$, and a target bit t for any $j \in [m]$. For any $j, h \in [m]$, G_{jh} is a generalized 6-CNOT gate with control bits x_{j1}, x_{j2}, $x_{j3}, x'_{h1}, x'_{h2}, x'_{h3}$, and a target bit t. A control bit $x_{jl}[x'_{hl}]$ is positive in G_{jh} if and only if $x_{jl}[x'_{hl}]$ is positive in $G_j[G'_h]$. We construct a $(2n+1) \times (2n+1)$ generalized 6-CNOT circuit $C(\phi)$ which is a cascade consisting of m^2 gates G_{jh} $(j, h \in [m])$. As an example, $C(\psi)$ for the Boolean function ψ defined in (5) is shown in Fig. 2(c). We have the following by Lemma 1.

Lemma 2. $G_{jh}\big((\boldsymbol{x}, \boldsymbol{x}', t)\big) = \Big(\boldsymbol{x}, \boldsymbol{x}', \Big(\overline{\rho_j(\boldsymbol{x})} \wedge \overline{\rho_h(\boldsymbol{x}')}\Big) \oplus t\Big)$. □

Lemma 2 implies that G_{jh} changes the target bit if and only if $\rho_j(\boldsymbol{x}) = 0$ and $\rho_h(\boldsymbol{x}') = 0$.

We now show that ϕ is satisfiable if and only if $\tau(C(\phi)) \leq 2$. For a gate G of C, $G[\boldsymbol{v}]$ is the output vector of G generated by an input vector $\boldsymbol{v}$ of C. Also, $w[\boldsymbol{v}]$ is the value of a wire w in C generated by $\boldsymbol{v}$.

Lemma 3. *A test set $T = \{\boldsymbol{v}_1, \boldsymbol{v}_2\}$ of a generalized CNOT circuit C with no 0-CNOT gate is complete if and only if T satisfies the following conditions:*

(i) $\boldsymbol{v}_2 = \overline{\boldsymbol{v}_1}$, *and*

(ii) $G[\boldsymbol{v}_i] = \boldsymbol{v}_i$ *($i \in [2]$) for every gate G of C.*

Proof. It is easy to see that if T satisfies (i) and (ii), then $W(C)$ is controllable by T. Thus T is complete for C by Theorem I.

Suppose T is complete for C. Then $W(C)$ is controllable by T by Theorem I. Since the input wires of C are controllable by T, we have $\boldsymbol{v}_2 = \overline{\boldsymbol{v}_1}$. Thus, T satisfies (i). Suppose T does not satisfy (ii), that is $G[\boldsymbol{v}_i] \neq \boldsymbol{v}_i$ for some generalized k-CNOT gate G and some i, say $i = 1$. That is, if $w_{\rm ti}$ and $w_{\rm to}$ are the input and

output wires of the target bit of G, we have

$$w_{\mathrm{to}}[\boldsymbol{v}_1] = \overline{w_{\mathrm{ti}}[\boldsymbol{v}_1]}. \tag{6}$$

Since the input wires of G are controllable by T, we have

$$w_{\mathrm{in}}[\boldsymbol{v}_2] = \overline{w_{\mathrm{in}}[\boldsymbol{v}_1]} \tag{7}$$

for every input wire w_{in} of G. Thus we conclude that

$$w_{\mathrm{ti}}[\boldsymbol{v}_2] = \overline{w_{\mathrm{ti}}[\boldsymbol{v}_1]}. \tag{8}$$

By (6), (7), and $k \geq 1$, there exists an input wire w_{in} of control bit of G such that $w_{\mathrm{in}}[\boldsymbol{v}_2] = 1$ if w_{in} is a negative control bit, and $w_{\mathrm{in}}[\boldsymbol{v}_2] = 0$ otherwise. This implies that

$$w_{\mathrm{to}}[\boldsymbol{v}_2] = w_{\mathrm{ti}}[\boldsymbol{v}_2]. \tag{9}$$

By (6), (8), and (9), we have

$$w_{\mathrm{to}}[\boldsymbol{v}_1] = w_{\mathrm{to}}[\boldsymbol{v}_2],$$

which means that w_{to} is not controllable by T, a contradiction. Thus T satisfies (ii). □

Now, we are ready to prove the following.

Lemma 4. *ϕ is satisfiable if and only if $\tau(C(\phi)) \leq 2$.*

Proof. It is easy to see from Lemmas 2 and 3 that if $\phi(\boldsymbol{x}) = 1$ for some $\boldsymbol{x} \in \{0,1\}^n$, then a test set $\{(\boldsymbol{x}, \overline{\boldsymbol{x}}, 0), (\overline{\boldsymbol{x}}, \boldsymbol{x}, 1)\}$ is complete for $C(\phi)$. Thus, $\tau(C(\phi)) \leq 2$.

Notice that $\tau(C) \geq 2$ for any reversible circuit C by Theorem I. Suppose $\tau(C(\phi)) = 2$, and let T be a complete test set of size two. By Lemma 3, $T = \{(\boldsymbol{x}, \boldsymbol{y}, 0), (\overline{\boldsymbol{x}}, \overline{\boldsymbol{y}}, 1)\}$ for some $\boldsymbol{x}, \boldsymbol{y} \in \{0,1\}^n$. Also by Lemma 3, $G_{jh}[(\boldsymbol{x}, \boldsymbol{y}, 0)] = (\boldsymbol{x}, \boldsymbol{y}, 0)$ and $G_{jh}[(\overline{\boldsymbol{x}}, \overline{\boldsymbol{y}}, 1)] = (\overline{\boldsymbol{x}}, \overline{\boldsymbol{y}}, 1)$ for any $j, k \in [m]$. Thus by Lemma 2,

$$\overline{\rho_j(\boldsymbol{x})} \wedge \overline{\rho_h(\boldsymbol{y})} = 0 \text{ and } \overline{\rho_j(\overline{\boldsymbol{x}})} \wedge \overline{\rho_h(\overline{\boldsymbol{y}})} = 0$$

for any $j, h \in [m]$, that is,

$$\rho_j(\boldsymbol{x}) \vee \rho_h(\boldsymbol{y}) = 1 \text{ and } \rho_j(\overline{\boldsymbol{x}}) \vee \rho_h(\overline{\boldsymbol{y}}) = 1$$

for any $j, h \in [m]$. If $\rho_j(\boldsymbol{x}) = 1$ for any $j \in [m]$, then $\phi(\boldsymbol{x}) = 1$, and ϕ is satisfiable. If $\rho_j(\boldsymbol{x}) = 0$ for some $j \in [m]$, then $\rho_h(\boldsymbol{y}) = 1$ for any $h \in [m]$. Thus $\phi(\boldsymbol{y}) = 1$, and ϕ is satisfiable. □

Since $C(\phi)$ can be constructed in polynomial time, we complete the proof of the theorem. □

4 Lower Bounds for 1-CNOT Circuits

The purpose of this section is to prove the following:

Theorem 2. *There exists a* 1*-CNOT circuit* C *such that*

$$\tau(C) = \Omega(\log\log|W(C)|).$$ □

Before proving the theorem, we need some preliminaries.

4.1 Preliminaries

The level of a wire of a reversible circuit is defined as follows. The input wires of the circuit are at level 0, and the output wires of a gate are at one plus the highest level of any of input wires of the gate. In cases where an input wire of a gate is at level i and the output wires are at level $j > i+1$, we say the input wire is at all levels between i and $j-1$ inclusively.

It is easy to see the following lemmas.

Lemma 5. *If* C_3 *is a reversible* 2×2 *circuit consisting of just one* 1*-CNOT gate, then* $\tau(C_3) = 3$. □

Lemma 6. *If* B *is a* 2×2 1*-CNOT circuit shown in Fig. 3, then* $B(\boldsymbol{v}) = \boldsymbol{v}$ *for any* $\boldsymbol{v} \in \{0,1\}^2$. □

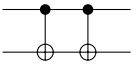


Fig. 3. 2×2 1-CNOT circuit B

Lemma 7. *If* C *is an* $n \times n$ 1*-CNOT circuit with* g *gates, then* $|W(C)| = n+2g$. □

4.2 Proof of Theorem 2

We prove the theorem by constructing such circuits. Let C_h ($h \geq 3$) be a 1-CNOT circuit defined as follows. Let C_3 be a 1-CNOT circuit consisting of just one 1-CNOT gate. For $h \geq 4$, C_h is recursively defined as follows. Let $C_{h-1}^{(0)}, C_{h-1}^{(1)}, \ldots, C_{h-1}^{(\varpi_{h-1})}$ be $\varpi_{h-1}+1$ copies of C_{h-1}, where $\varpi_{h-1} = |W(C_{h-1})|$. Construct an $n_{h-1} \times n_{h-1}$ 1-CNOT circuit D_{h-1} by concatenating $C_{h-1}^{(1)}, C_{h-1}^{(2)}, \ldots, C_{h-1}^{(\varpi_{h-1})}$, where n_{h-1} is the number of input wires of C_{h-1}. Let $W(C_{h-1}^{(k)}) = \{w_1^{(k)}, w_2^{(k)}, \ldots, w_{\varpi_{h-1}}^{(k)}\}$ for $0 \leq k \leq \varpi_{h-1}$ such that if the level of $w_i^{(k)}$ is not greater than the level of $w_j^{(k)}$, then $i \leq j$. C_h is constructed from D_{h-1} and $C_{h-1}^{(0)}$ by inserting a copy of 1-CNOT circuit B shown in Fig. 3 for each wire of $C_{h-1}^{(i)}$, $i \in [\varpi_{h-1}]$, such that the wire of $C_{h-1}^{(i)}$ is the control bit and $w_i^{(0)}$is the target bit of the 1-CNOT gates. As an example, D_3 and C_4 are shown in Fig. 4 and Fig. 5, respectively.

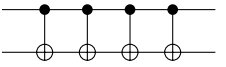


Fig. 4. 1-CNOT circuit D_3

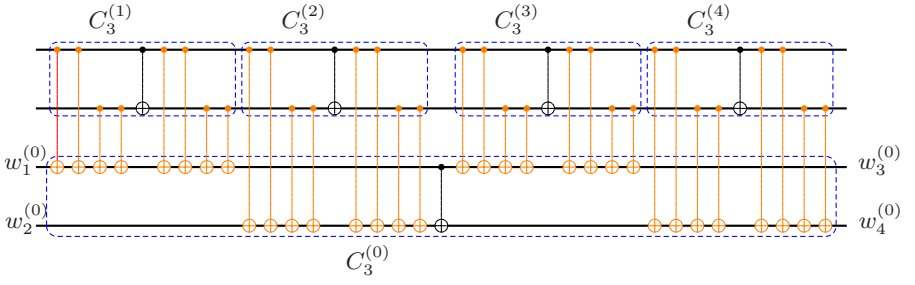


Fig. 5. 4×4 1-CNOT circuit C_4

Let g_h be the number of gates in C_h. From the definition of C_h, we have

$$n_h = 2^{h-2} \tag{10}$$

for $h \geq 3$. We also have

$$\begin{aligned} g_h &= \big(\varpi_{h-1} + 1\big) g_{h-1} + 2\varpi_{h-1}^2 \\ &= \big(n_{h-1} + 2g_{h-1} + 1\big) g_{h-1} + 2(n_{h-1} + 2g_{h-1})^2 \qquad (11) \\ &= 10g_{h-1}^2 + g_{h-1}(9n_{h-1} + 1) + 2n_{h-1}^2 \qquad (12) \end{aligned}$$

for $h \geq 4$, where (11) follows from Lemma 7. Since each input wire of C_h is an input wire of a gate, and every 1-CNOT gate has two input wires, we have

$$n_h \leq 2g_h \tag{13}$$

for $h \geq 3$.

Lemma 8. $h = \Omega(\log\log|W(C_h)|)$.

Proof. By (12) and (13), we have

$$g_h \leq 36g_{h-1}^2 + g_{h-1} \leq 37g_{h-1}^2.$$

It follows that $37g_h \leq (37g_{h-1})^2$, and so

$$\log g_h + \log 37 \leq 2(\log g_{h-1} + \log 37) \leq 2^{h-3}(\log g_3 + \log 37).$$

Thus, we have

$$\log g_h \leq 2^{h-3}(0 + \log 37). \tag{14}$$

By Lemma 7 and (13), we have $\varpi_h = n_h + 2g_h \leq 4g_h$ for $h \geq 4$, and so

$$\log\log \varpi_h \leq \log\log g_h + 1 \leq h - 3 + \log\log 37 + 1$$

by (14). Thus we conclude that

$$h = \Omega(\log\log \varpi_h).$$

□

Lemma 9. $\tau(C_h) \geq h$.

Proof. The proof is by induction on h. $\tau(C_3) = 3$ by Lemma 5. Suppose $\tau(C_{h-1}) \geq h-1$. We will show that $\tau(C_h) \geq h$. Suppose contrary that $\tau(C_h) = h-1$. Let $T = \{\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_{h-1}\}$ be a complete test set for C_h, and $\boldsymbol{v}_l = (\boldsymbol{v}_l^{(1)}, \boldsymbol{v}_l^{(2)})$ for $\boldsymbol{v}_l^{(1)}, \boldsymbol{v}_l^{(2)} \in \{0,1\}^{n_{h-1}}$ ($l \in [h-1]$). Let $T' = \{\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_{h-2}\}$, and $T'_k = \{\boldsymbol{v}_1^{(k)}, \boldsymbol{v}_2^{(k)}, \ldots, \boldsymbol{v}_{h-2}^{(k)}\}$ for $k \in [2]$.

Since $\tau(C_{h-1}) \geq h-1$ by the inductive hypothesis, $W(C_{h-1})$ is not controllable by T'_k, $k \in [2]$. Thus there exists i such that $w_i^{(0)}$ in $C_{h-1}^{(0)}$ is not controllable by T'_2. There also exists j such that $w_j^{(i)}$ in D_{h-1} is not controllable by T'_1. Thus, we have

$$(w_j^{(i)}[\boldsymbol{v}_l^{(1)}], w_i^{(0)}[\boldsymbol{v}_l^{(2)}]) = (w_j^{(i)}[\boldsymbol{v}_m^{(1)}], w_i^{(0)}[\boldsymbol{v}_m^{(2)}]) \tag{15}$$

for any $\boldsymbol{v}_l, \boldsymbol{v}_m \in T'$.

Let G be the left 1-CNOT gate of a copy of B whose control bit is at $w_j^{(i)}$ and target bit is at $w_i^{(0)}$, w_c be the input wire of control bit of G, and w_t be the input wire of target bit of G. Then by Lemma 6,

$$(w_c[\boldsymbol{v}_l], w_t[\boldsymbol{v}_l]) = (w_j^{(i)}[\boldsymbol{v}_l^{(1)}], w_i^{(0)}[\boldsymbol{v}_l^{(2)}]) \tag{16}$$

for any $\boldsymbol{v}_l \in T'$. By (15) and (16), we have

$$(w_c[\boldsymbol{v}_l], w_t[\boldsymbol{v}_l]) = (w_c[\boldsymbol{v}_m], w_t[\boldsymbol{v}_m]) \tag{17}$$

for any $\boldsymbol{v}_l, \boldsymbol{v}_m \in T'$. By Lemma 5 and (17), we conclude that $W(G)$ is not controllable by $T = T' \cup \{\boldsymbol{v}_{h-1}\}$, a contradiction. Thus, we have $\tau(C_h) \geq h$. □

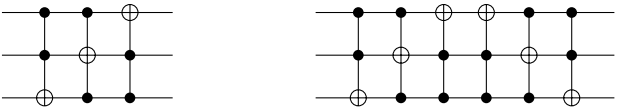
From Lemma 8 and 9, we obtain the theorem.

5 Lower Bounds for 2-CNOT Circuits

The purpose of this section is to prove the following.

Theorem 3. *There exists an $n \times n$ 2-CNOT circuit C such that $\tau(C) = \Omega(\log n)$.*

Proof. We need the following two lemmas, which can be easily seen.



(a) 3×3 2-CNOT circuit E_3. (b) 3×3 2-CNOT circuit F.

Fig. 6. 3×3 2-CNOT circuits E_3 and F

Lemma 10. *If E_3 is a 3×3 2-CNOT circuit shown in Fig. 6(a), then $\tau(E_3) = 3$.* □

Lemma 11. *If F is a 3×3 2-CNOT circuit shown in Fig. 6(b), then $F(\boldsymbol{v}) = \boldsymbol{v}$ for any $\boldsymbol{v} \in \{0, 1\}^3$.* □

We prove the theorem by constructing such circuits. Let E_h ($h \geq 3$) be a 2-CNOT circuit defined as follows. Let E_3 be a 2-CNOT circuit shown in Fig. 6(a). For $h \geq 4$, E_h is recursively defined as follows. Let $E_{h-1}^{(i)}$ for $0 \leq i \leq \varpi_{h-1}$ and $E_{h-1}^{(j,k)}$ for $j, k \in [\varpi_{h-1}]$ be copies of E_{h-1}, where $\varpi_{h-1} = |W(E_{h-1})|$. Construct an $n_{h-1} \times n_{h-1}$ 2-CNOT circuit H_{h-1} by concatenating $E_{h-1}^{(1)}, E_{h-1}^{(2)}, \ldots, E_{h-1}^{(\varpi_{h-1})}$, and construct an $n_{h-1} \times n_{h-1}$ 2-CNOT circuit J_{h-1} by concatenating $E_1^{(1,1)}, E_1^{(1,2)}, \ldots, E_1^{(1,\varpi_{h-1})}, E_2^{(2,1)}, E_2^{(2,2)}, \ldots, E_2^{(2,\varpi_{h-1})}, \ldots, E_{\varpi_{h-1}}^{(\varpi_{h-1},\varpi_{h-1})}$,
where n_{h-1} is the number of input wires of E_{h-1}. Let $W(E_{h-1}^{(i)}) = \{w_1^{(i)}, w_2^{(i)}, \ldots, w_{\varpi_{h-1}}^{(i)}\}$ and $W(E_{h-1}^{(j,k)}) = \{w_1^{(j,k)}, w_2^{(j,k)}, \ldots, w_{\varpi_{h-1}}^{(j,k)}\}$ such that if the level of $w_i^{(*)}$ is not greater than the level of $w_j^{(*)}$, then $i \leq j$. E_h is constructed from J_{h-1}, H_{h-1}, and $E_{h-1}^{(0)}$ by inserting a copy of F for each wire $w_k^{(i,j)}$ with $i, j, k \in [\varpi_{h-1}]$ such that $w_k^{(i,j)}$ of $E_{h-1}^{(i,j)}$ in J_{h-1} is the top bit of the copy of F, $w_j^{(i)}$ of $E_{h-1}^{(i)}$ in H_{h-1} is the middle bit of the copy of F, and $w_i^{(0)}$ of $E_{h-1}^{(0)}$ is the bottom bit of the copy of F.

From the definition of E_h, we have $n_h = 3^{h-2}$, and so the following.

Lemma 12. $h = \Omega(\log n_h)$. □

Lemma 13. $\tau(E_h) \geq h$.

Proof (Sketch). The proof is by induction on h. $\tau(E_3) = 3$ by Lemma 10. Suppose $\tau(E_{h-1}) \geq h - 1$. We will show that $\tau(E_h) \geq h$. Suppose contrary that $\tau(E_h) = h - 1$, and let $T = \{\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_{h-1}\}$ be a complete test set for E_h. Since $\tau(E_{h-1}) \geq h - 1$, $W(E_{h-1})$ is not controllable by $T' = \{\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots \boldsymbol{v}_{h-2}\}$. Thus there exist $i, j, k \in [\varpi_{h-1}]$ such that none of $w_i^{(0)}$, $w_j^{(i)}$, and $w_k^{(i,j)}$ is controllable by T'. It follows that if $F_{i,j,k}$ is a copy of F with the top bit on $w_k^{(i,j)}$, and $E_{i,j,k}$ is a copy of E consisting of the left three gates of $F_{i,j,k}$, then $W(E_{i,j,k})$ is not controllable by Lemmas 10 and 11, a contradiction. Thus, we have $\tau(E_h) \geq h$. □

From Lemmas 12 and 13, we obtain the theorem. □

6 Concluding Remarks

It should be noted that (1) is merely an existential upper bound. It is an interesting open problem to find a polynomial time algorithm to construct a complete test set of such size.

We can show that $\tau(E_h) = \Omega(\log\log|W(E_h)|)$, though the proof is rather complicated.

References

1. Bennett, C.: Logical reversiblity of computation. IBM J. Res. Dev. 525–532 (1973)
2. Chakraborty, A.: Synthesis of reversible circuits for testing with universal test set and c-testability of reversible iterative logic array. In: Proc. of the 18th International Conference on VLSI Design (2005)
3. Merkle, R.: Two types of mechanical reversible logic. Nanotechnology, 114–131 (1993)
4. Nielsen, M., Chuang, I.: Quantum Computation and Quantum Information. Cambridge University Press, Cambridge (2000)
5. Patel, K., Hayes, J., Markov, I.: Fault testing for reversible circuits. IEEE Trans. Computer-Aided Design, 1220–1230 (2004)
6. Shende, V., Prasad, A., Markov, I., Hayes, J.: Synthesis of reversible logic circuits. IEEE Trans. Computer-Aided Design, 710–722 (2003)

The Space Complexity of k-Tree Isomorphism

V. Arvind[1], Bireswar Das[1], and Johannes Köbler[2]

[1] The Institute of Mathematical Sciences, Chennai 600 113, India
{arvind,bireswar}@imsc.res.in
[2] Institut für Informatik, Humboldt Universität zu Berlin, Germany
koebler@informatik.hu-berlin.de

Abstract. We show that isomorphism testing of k-trees is in the class StUSPACE($\log n$) (strongly unambiguous logspace). This bound follows from a deterministic logspace algorithm that accesses a strongly unambiguous logspace oracle for canonizing k-trees. Further we give a logspace canonization algorithm for k-paths.

1 Introduction

It often turns out that NP-hard graph problems when restricted to the class of *partial k-trees* for constant k have efficient polynomial-time algorithms [Bod88, SS87]. Partial k-trees are also known as the class of graphs of treewidth k. For constant k, in general, they are known as *bounded treewidth graphs* (formal definitions are given in Section 3).

Graph Isomorphism is the problem of deciding whether two given graphs are isomorphic. I.e. the problem is to test whether there is a bijective function that maps vertices of the first graph to vertices of the second graph and preserves the edge relation. Graph Isomorphism has attracted much algorithmic research. It is one of the few problems in NP that is neither known to be computable in polynomial time nor to be NP-complete. Polynomial time algorithms have been designed for the problem for several interesting restricted graph classes [Luk82, Mil83, Bab86], including the class of partial k-trees [Bod90]. Bodlaender [Bod90] gave the first polynomial-time algorithm for testing the isomorphism of partial k-trees. Bodlaender's algorithm, based on dynamic programming, runs in time $O(n^{k+4.5})$.

Our interest is in a *complexity-theoretic* characterization of Graph Isomorphism for partial k-trees using space bounded complexity classes. We explain our motivation for studying the space complexity of k-tree isomorphism. On the one hand, we have Lindell's result [Lin92, JKMT03] that tree canonization is complete for deterministic logspace,[1] which tightly characterizes the complexity of both isomorphism and canonization of trees. What about partial k-tree isomorphism? The recent TC^1 upper bound for isomorphism of partial k-trees by Grohe and Verbitsky [GV06] raises the question about a tight complexity-theoretic classification of the problem. It is tempting to conjecture that partial

[1] Provided that the tree is given in the pointer notation; using the parenthesis notation the problem is NC^1-complete [Bus97, JKMT03]

T. Tokuyama (Ed.): ISAAC 2007, LNCS 4835, pp. 822–833, 2007.

k-tree isomorphism should also be complete for deterministic logspace, just like ordinary tree isomorphism. However, the best known complexity bound for even recognizing partial k-trees is LOGCFL (the class of decision problems that are logspace many-one reducible to CFLs) [Wan94].

The TC^1 bound of [GV06] suggests that we can put the problem in a natural complexity class contained in TC^1 like LOGCFL or DET, or perhaps somewhere in the logspace counting hierarchy. The logspace counting classes, introduced in the seminal paper [AO96], contain many natural problems sitting in NC^2 and have been used to characterize most natural problems in NC^2 satisfactorily from a complexity-theoretic viewpoint. A comprehensive study can be found in Allender's survey article [All04].

In this paper we show that *full* k-tree canonization is in FL^{NL}. Recall that the *canonization problem* for graphs is to produce a *canonical form* $canon(G)$ for a given graph G such that $canon(G)$ is isomorphic to G and $canon(G_1) = canon(G_2)$ for any pair of isomorphic graphs G_1 and G_2. Canonization is clearly at least as hard as Graph Isomorphism. In fact, it is easy to see that Graph Isomorphism is even AC^0 reducible to Graph Canonization. However, in general it is not known if the two problems are even polynomial-time equivalent.

Interestingly, the NL oracle required for k-tree canonization is a language computed by an NL machine M that is *strongly unambiguous*: for any two configurations x and y of machine M there is at most one computation path from x to y. The class of languages accepted by such NL machines is denoted $StUSPACE(\log n)$ (StUL for short) by Allender and Lange [AL89]. As shown in [AL89], StUL is in fact contained in $DSPACE(\log^2 n/\log\log n)$, improving the $DSPACE(\log^2 n)$ bound given by Savitch's theorem. Furthermore, the complexity class StUL is closed under complementation and even closed under logspace Turing reductions [BJLR91, Corollary 15]. Thus, it follows that k-tree isomorphism is in StUL. The class StUL is not known to be contained in L. In fact, it contains the class ULIN of unambiguous linear languages [BJLR91] which is not known to be in L. To the best of our knowledge, no explicit example of a language in StUL is known that is not already in L. Thus, k-tree isomorphism is the first natural problem in StUL which is not known to be in L.

We note that parallel algorithms are known for k-tree isomorphism. For example, in [GSS02] a processor efficient AC^2 algorithm was given for k-tree isomorphism. Since $StUL \subseteq UL \subseteq NL \subseteq AC^1$, our upper bound is tighter than previously known bounds from a complexity-theoretic perspective. We also look into the problem of canonizing k-paths, a special case of k-trees, and give a logspace canonization algorithm for k-paths.

2 Preliminaries

By graphs we mean finite simple graphs. For basic graph theoretic definitions we refer the reader to [Die97]. For a graph G, let $V(G)$ denote its vertex set and $E(G)$ denote its edge set. The set $\{w \in V(G) \mid \{v, w\} \in E(G)\}$ of all *neighbors* of

$v \in V(G)$ is denoted by $N(v)$. For a subset $U \subseteq V(G)$, we use $G[U]$ to denote the *induced subgraph* of G, where $V(G[U]) = U$ and $E(G[U]) = \{e \in E(G) \mid e \subseteq U\}$.

Two graphs G and H are isomorphic if there is an edge-preserving bijection $\tau : V(G) \to V(H)$, i.e., for all $u, v \in V(G)$, $\{u, v\} \in E(G)$ if and only if $\{\tau(u), \tau(v)\} \in E(H)$. In case the vertices of G and H are labeled, then the isomorphism τ must also preserve the labels.

Next we recall some complexity classes defined by circuits and some space bounded classes.

A language A is in the complexity class NC^i (resp. AC^i) if there is a uniform family of circuits $\{C_n\}_n$ of depth $O(\log^i n)$ and size $poly(n)$ with internal AND, OR and NOT gates with bounded (resp. unbounded) fan-in that accepts A. TC^i is the extension of AC^i where we additionally allow unbounded $MAJORITY$ gates.

The complexity class L consists of all languages A accepted by a deterministic $O(\log n)$ space bounded Turing machine. NL is defined in the same way by using nondeterministic machines. FL is the class of all functions computable by a deterministic $O(\log n)$ space bounded Turing machine.

A nondeterministic Turing machine M is called *unambiguous*, if for any input x, it has at most one accepting computation path.

M is said to be *reach-unambiguous* if it is unambiguous and for any input x, there is at most one computation path from the starting configuration to any other configuration.

M is said to be *strongly unambiguous* if it is unambiguous and for any pair of configurations u and v of M there is at most one computation path from u to v.

A *mangrove* is a directed acyclic graph such that there is at most one directed path from i to j for any two nodes i and j in the graph. In other words, a directed graph is a mangrove if and only if for any node u the subgraph induced by u and all nodes reachable from u is a rooted directed tree.

Note that an unambiguous machine M is strongly unambiguous if and only if its configuration graph is a mangrove.

A language A is in the class UL (RUL, StUL) if there is an $O(\log n)$ space bounded unambiguous (reach-unambiguous, strongly unambiguous, respectively) Turing machine accepting A. It is well known that

$$\mathrm{NC}^1 \subseteq \mathrm{L} \subseteq \mathrm{StUL} \subseteq \mathrm{RUL} \subseteq \mathrm{UL} \subseteq \mathrm{NL} \subseteq \mathrm{AC}^1 \subseteq \mathrm{TC}^1 \subseteq \mathrm{NC}^2.$$

The following result of Allender and Lange [AL89] shows that Savitch's $\log^2 n$ space deterministic simulation of NL can be improved for StUL and RUL.

Theorem 1. [AL89] $\mathrm{StUL} \subseteq \mathrm{RUL} \subseteq \mathrm{DSPACE}(\log^2 n / \log\log n)$.

3 *k*-Tree Canonization

Let $\mathcal{G}$ be a class of (encodings of) graphs. We say that f computes a *complete invariant* for $\mathcal{G}$, if

$$\forall G, H \in \mathcal{G} : G \cong H \Leftrightarrow f(G) = f(H).$$

A complete invariant f for $\mathcal{G}$ that computes for any graph $G \in \mathcal{G}$ a graph $f(G)$ that is isomorphic to G is called a *canonization* for $\mathcal{G}$. We call $f(G)$ the *canon* of G (w.r.t. f).

Notice that if there is a polynomial time computable canonization for $\mathcal{G}$ then the graph isomorphism problem restricted to $\mathcal{G}$ can also be solved in polynomial time. As shown by Gurevich, canonization of general graphs is polynomial-time equivalent to computing a complete invariant [Gur97].

Certain complete invariants are known to be intractable. For example, it is NP-hard to compute the lexicographically least graph (w.r.t. some specific representation) isomorphic to the given graph [BL83]. However, the approach of computing complete invariants has been successful for solving the graph isomorphism problem efficiently for some graph classes [BL83, Spi96].

The classes of k-trees and partial k-trees were introduced by Arnborg and Proskurowski (see e.g. [AP89]).

Definition 2. *The class of* k-trees *is inductively defined as follows.*

- *A clique with k vertices (k-clique for short) is a k-tree.*
- *Given a k-tree G' with n vertices, a k-tree G with $n+1$ vertices can be constructed by introducing a new vertex v and picking a k-clique C in G' and then joining v to each vertex u in C. Thus, $V(G) = V(G') \cup \{v\}$, $E(G) = E(G') \cup \{\{u, v\} \mid u \in C\}$.*

A partial k-tree *is a subgraph of a k-tree.*

Before we go into the k-tree canonization we notice that the following characterization of k-trees gives a logspace algorithm for recognizing k-trees.

Definition 3. [Klo94] *Let $G = (V, E)$ be a graph. A subset S of V is called a* vertex separator *for two nonadjacent vertices $u, v \in V$, if in the subgraph of G induced by the vertex set $V - S$ the two vertices u, v are in different connected components. A vertex separator S for u, v is called* minimal*, if no proper subset of S is a vertex separator for u and v. A subset $S \subseteq V$ is a* minimal vertex separator *if S is a minimal vertex separator for some pair of vertices $u, v \in V$.*

Lemma 4. [CI88] *A graph G with $n > k$ vertices is a k-tree if and only if*

- *every pair of nonadjacent vertices u and v has a k-clique as a minimal vertex separator and*
- *$E(G)$ contains exactly $\binom{k}{2} + k(n-k)$ edges.*

It is easy to see that the two conditions of Lemma 4 can be checked in logspace. Hence, from now on we can assume that the input graph G is a k-tree. Further we assume that $V(G) = \{1, \ldots, n\}$.

Our algorithm for k-tree canonization works by reducing the problem to the problem of canonizing certain labeled trees that encode essential information about k-trees. Our initial goal is to define this labeled tree. For this we use the concept of the layer decomposition of a k-tree with respect to a base B. This

concept was introduced in [KCP82] for testing isomorphism in hookup classes. Subsequently, it was used in [Cha90, GSS02] for the design of efficient k-tree isomorphism algorithms.

Definition 5. (cf. [KCP82, GSS02]) *Let $G = (V, E)$ be a k-tree and let $B \subseteq V$ be a k-clique in G. Then the* B-decomposition *of G is the sequence of sets $B(0), \ldots, B(p)$ such that $B(0) = B$ and $p = \max\{i \geq 0 \mid B(i) \neq \emptyset\}$, where $B(i+1)$ is inductively defined by*

$$B(i+1) = \{v \in V - B[i] \mid N(v) \cap B[i] \text{ is a } k\text{-clique}\}.$$

Here, $B[i]$ denotes the union $B[i] = B(0) \cup \cdots \cup B(i)$.

The set $B(i)$ is called the ith layer of the B-decomposition of G. Intuitively, the layers of the B-decomposition indicate the order in which vertices could be added to G when we choose B as the initial k-clique. More precisely, starting with the k-tree $G_0 = G[B]$, $G_{i+1} = G[B[i+1]]$ can be constructed from $G_i = G[B[i]]$ by adding the vertices in $B(i+1)$ to G_i. Recall that v can be added to G_i if and only if the set $N(v) \cap B[i]$ of v's neighbors in $B[i]$, henceforth denoted by $N_i(v)$, induces a k-clique in G_i. In [KCP82], this set $N_i(v)$ is called the *support* of $v \in B(i)$).

If this process is successful, i.e., if each vertices of G is covered by some layer $B(i)$, then B is called a *base* of G (cf. [KCP82]).

We first show that any k-clique B in G can be used as a base for constructing G (see Lemma 9).

Definition 6. (cf. [GSS02]) *A vertex v of a k-tree G is called* simplicial, *if $N(v)$ induces a k-clique in G.*

Claim 7. *Any k-tree G with $n \geq k+2$ vertices has at least two nonadjacent simplicial vertices.*

Proof. The proof is by induction on n. If $n = k+2$, then G is obtained from a $(k+1)$-clique G' by choosing a k-clique C in G' and introducing a new node v which is joined to each vertex in C. Let u be the unique vertex in G' not covered by C. Then u and v are two nonadjacent simplicial vertices in G. For the inductive step assume that G has $n > k+2$ vertices. Then G has been obtained from a k-tree G' by introducing a new node v and joining it to each vertex in a k-clique C of G'. Clearly, v is simplicial in G. Further, by the induction hypothesis, G' has two nonadjacent simplicial vertices u_1 and u_2. Since u_1 and u_2 are nonadjacent, at least one of them does not belong to C and therefore it is also simplicial in G. □

Claim 8. *Let B be a k-clique of a k-tree G with $n \geq k+1$ vertices. Then G has a simplicial vertex $v \notin B$.*

Proof. If $n = k+1$, then the only vertex not in B is simplicial. If $n > k+1$, then Claim 7 guarantees the existence of two nonadjacent simplicial vertices that cannot be both in B. □

Lemma 9. *For any k-tree $G = (V, E)$ and any k-clique B, the B-decomposition forms a partition of V.*

Proof. The proof is by induction on n. The base case $n = k$ is clear. For the inductive step assume that $n \geq k+1$ and let $B(0), \ldots, B(p)$ be the B-decomposition of G. By Claim 8, G has a simplicial vertex v not in B. It is easy to prove that $G - v$ is a k-tree and hence, by the induction hypothesis, the B-decomposition $B'(0), \ldots, B'(p')$ of $G-v$ forms a partition of $V-\{v\}$. Now let $i \geq 0$ be the minimum integer such that $N(v) \subseteq B'[i]$. Then it follows that $B(i+1) = B'(i+1) \cup \{v\}$ and $B(j) = B'(j)$ for all $j \neq i+1$, implying that $V = B[p]$. □

The following properties of the B-decomposition have been proved in [KCP82].

Proposition 10. *If B is a base for a k-tree $G = (V, E)$, then the B-decomposition $B(0), \ldots, B(p)$ has the following properties.*

1. *Any two vertices in $B(i)$, $i \geq 1$, are nonadjacent. Hence, $N_{i-1}(v) = N_i(v)$ for any vertex $v \in B(i)$.*
2. *Any vertex $v \in B(i)$, $i \geq 2$, has a unique neighbor $f(v) \in B(i-1)$, called the* father *of v w.r.t. B.*

In order to efficiently compute information on the B-decomposition of a k-tree G we use a directed graph $D(G, B)$ which is defined as follows (whenever G and B are clear from the context we simply write D instead of $D(G, B)$). D has the vertex set

$$V(D) = \{B\} \cup \{(C, v) \mid v \notin C \text{ and } C \cup \{v\} \text{ is a } (k+1)\text{-clique in } G\}$$

and the vertices of D are joined by the directed edges in the set

$$\begin{aligned} E(D) = &\{(B, (B, v)) \mid (B, v) \in V(D)\} \cup \\ &\{((C, v), (C', v')) \mid v \in C', v' \notin C, \|C \cap C'\| = k-1\}. \end{aligned}$$

This means that in D we provide a transition from (C, v) to (C', v') if C' can be obtained from C by replacing some vertex $u \in C$ by v, i.e., $C' = (C - \{u\}) \cup \{v\}$. Our next aim is to show that D is a mangrove (see Lemma 13).

Claim 11. *For any vertex $v \in B(i)$, $i \geq 1$, D has a directed path of length i from B to $(N_{i-1}(v), v)$.*

Proof. We prove the claim by induction on i. The base case $i = 1$ is clear. For the inductive step assume that $v \in B(i)$, $i \geq 2$ and let $f(v) \in B(i-1)$ be the father of v. By the induction hypothesis it follows that D has a directed path of length $i-1$ from B to $(N_{i-2}(f(v)), f(v))$. Clearly, $f(v) \in N_{i-1}(v)$ and $v \notin N_{i-2}(f(v))$. Further, since $f(v)$ is the only vertex in $N_{i-1}(v)$ belonging to $B(i-1)$, the remaining $k-1$ vertices belong to $B[i-2]$ and, as they are also neighbors of $f(v)$, they belong to the support $N_{i-2}(f(v))$ of $f(v)$. This shows that D has an edge from $(N_{i-2}(f(v)), f(v))$ to $(N_{i-1}(v), v)$. □

Claim 12. *If D has a directed path $B, (C_1, v_1), \ldots, (C_{i-1}, v_{i-1}), (C, v)$ of length $i \geq 1$ from B to some vertex (C, v), then $v \in B(i)$ and $C = N_{i-1}(v) \subseteq B \cup \{v_1, \ldots, v_{i-1}\}$.*

Proof. Again the proof is by induction on i. If $E(D)$ contains the edge $(B, (B, v))$, then clearly $v \in B(1)$ via the support $N_0(v) = B$.

For the inductive step let us assume that $B, (C_1, v_1), \ldots, (C_{i-1}, v_{i-1}), (C, v)$ is a directed path of length $i \geq 2$ from B to (C, v). Then by the induction hypothesis it follows that $v_{i-1} \in B(i-1)$ via the support $C_{i-1} = N_{i-2}(v_{i-1}) \subseteq B \cup \{v_1, \ldots, v_{i-2}\}$. As $N_{i-2}(v_{i-1}) = N_{i-1}(v_{i-1})$ by part 1 of Proposition 10, this implies that C_{i-1} contains all neighbors of v_{i-1} in $B[i-1]$. Since v is a neighbor of v_{i-1} that does not belong to C_{i-1}, v cannot be in $B[i-1]$. As $((C_{i-1}, v_{i-1}), (C, v)) \in E(D)$, it follows that C is obtained from C_{i-1} by replacing some vertex u in C_{i-1} by v_{i-1}, i.e., $C = (C_{i-1} - \{u\}) \cup \{v_{i-1}\} \subseteq B \cup \{v_1, \ldots, v_{i-1}\}$. Hence, all vertices in C belong to $B[i-1]$ and are adjacent to v, implying that $v \in B(i)$ via the support $N_{i-1}(v) = C$ (notice that $C \subsetneq N_{i-1}(v)$ would imply $v \notin B[p]$). □

Lemma 13. *For any k-clique B in a k-tree G, the graph $D(G, B)$ is a mangrove.*

Proof. We first show that $D = D(G, B)$ does not have different paths from B to the same node (C, v).

By Claim 12, all paths from B to (C, v) have the same length. In order to derive a contradiction let i be minimal such that there are two different paths $B, (C_1, v_1), \ldots, (C_{i-1}, v_{i-1}), (C, v)$ and $B, (C'_1, v'_1), \ldots, (C'_{i-1}, v'_{i-1}), (C, v)$ of length i from B to some node (C, v). Then v_{i-1} and v'_{i-1} must be different, since otherwise Claim 11 implies that $C_{i-1} = N_{i-2}(v_{i-1}) = N_{i-2}(v'_{i-1}) = C'_{i-1}$, contradicting the minimality of the path length i. But now Claim 12 implies that v_{i-1} and v'_{i-1} both belong to $B(i-1)$ as well as to the support C of v, contradicting part 2 of Proposition 10.

To complete the proof suppose there are different directed paths between two nodes (C, v) and (C', v') in $D(G, B)$. Then we would also have different directed paths between the two nodes C and (C', v') in $D(G, C)$, contradicting the argument above. □

Now let $T = T(G, B)$ be the subgraph of $D(G, B)$ induced by the vertices reachable from B. Then Lemma 13 implies that T is a directed rooted tree with root B.

In fact, from Claims 11 and 12 it is immediate that by projecting the first component out from the nodes $(C, v) \in V(T)$ we get exactly the tree $T(G)$ defined in [KCP82]. There, the following labeling with respect to a bijection $\theta : B \to \{1, 2, \ldots, k\}$ has been defined.

Let (C, v) be a node in T. W.l.o.g. suppose that $C = \{v_1, \ldots, v_k\}$ where $C \cap B = \{v_1, \ldots, v_m\}$ for some $m \leq k$. Notice that by part 1 of Proposition 10, the $k - m$ vertices in $C - B$ belong to $k - m$ different layers $B(i_1), \ldots, B(i_{k-m})$. Then vertex (C, v) is labeled by the set $\{\theta(v_1), \ldots, \theta(v_m), k + i_1, \ldots, k + i_{k-m}\}$.

We denote the tree T together with this labeling by $T(G, B, \theta)$. The following theorem is due to [KCP82].

Theorem 14. *Let G and G' be two k-trees, let B be a base for G and let $\theta : B \to \{1, \ldots, k\}$ be a bijection. Then G and G' are isomorphic if and only if there exists a base B' for G' and a bijection $\theta' : B' \to \{1, \ldots, k\}$ such that the two labeled trees $T(G, B, \theta)$ and $T(G', B', \theta')$ are isomorphic.*

The proof of Theorem 14 crucially hinges on the fact that each isomorphic copy T' of the labeled tree $T(G, B, \theta)$ provides enough information to reconstruct G from T' up to isomorphism. To see why, for $i \geq 1$ let B_i be the set of vertices of T' that have distance i from the root of T' and let p be the maximum distance of any vertex in T' from the root. Then starting with a k-clique G_0 we can successively add in parallel all the vertices $v \in B_i$ to G_{i-1} for $i = 1, \ldots, p$. The crucial observation is that the labeling $\{\theta(v_1), \ldots, \theta(v_m), k + i_1, \ldots, k + i_{k-m}\}$ of the node v in T' tells us to which vertices in G_{i-1} vertex v should be connected (recall that Claim 12 guarantees that all vertices in the support of a node either belong to the base or lie on the path from the root to that vertex in the corresponding tree).

To canonize k-trees we use Lindell's [Lin92] deterministic logspace canonization algorithm for trees which can be made to work for any labeled tree by constructing gadgets for labels. More precisely, consider the algorithm A that on input a k-tree G computes the canon of all labeled trees $T(G, B, \theta)$ for all k-cliques B in G and all bijections $\theta : B \to \{1, \ldots, k\}$ and picks the lexicographically least among them. Then Theorem 14 implies that

- if two k-trees G and H are isomorphic then any tree of the form $T(G, B, \theta)$ is isomorphic to some tree of the form $T(H, B', \theta')$ and
- if G and H are non-isomorphic then no tree of the form $T(G, B, \theta)$ is isomorphic to some tree of the form $T(H, B', \theta')$.

Hence, A outputs the same tree T for both k-trees G and H if and only if G and H are isomorphic, implying that A computes a complete invariant for k-trees.

Furthermore, as explained above, the output tree T of A on input G provides enough information to reconstruct G from T in logspace up to isomorphism. The combination of A with this reconstruction procedure thus yields the desired canonization algorithm A' for k-trees. It remains to show that A can be implemented in $\mathrm{FL}^{\mathrm{StUL}}$. In the next lemma we show that the labeled trees $T(G, B, \theta)$ can be computed in logspace relative to some oracle in StUL. The following claim provides this oracle.

Claim 15. *The problem of deciding whether a vertex (C, v) of D has distance i from B is in* StUL.

Proof. The algorithm tries to guess a path of length i from B to (C, v) in the tree $T = T(G, B)$. For that, starting with vertex B, it iteratively guesses a next node (C', v') and checks if T provides an edge from the actual node to that node. If after i steps the algorithm reaches (C, v) then it accepts, otherwise it rejects.

Clearly, the algorithm runs in logspace since it has to store only two nodes of T and some counters. Since $D(G, B)$ is a mangrove by Claim 13, it is easy to see that the configuration graph is also a mangrove. □

Lemma 16. *On input a k-tree G, a k-clique B and a bijection $\theta : B \to \{1, \ldots, k\}$, the labeled tree $T(G, B, \theta)$ can be computed in logspace relative to some oracle in* StUL*.*

Proof. The algorithm for generating $T = T(G, B, \theta)$ first outputs $V(T)$ by checking for each node (C, v) in $V(D)$ whether it is reachable from B by using the StUL oracle of Claim 15. If so, it computes the label of (C, v) by recomputing the layer numbers of all the vertices in C (again using the StUL oracle). Finally, for each distinct pair of nodes in $V(T)$ it checks whether D provides a directed edge between them. □

This shows that the algorithm A described above can indeed be implemented in logspace relative to some oracle in StUL. Hence, we can state our main result.

Theorem 17. *For each fixed k there is a canonizing algorithm for k-trees that runs in* $\mathrm{FL}^{\mathrm{StUL}}$*.*

As StUL is closed under logspace Turing reductions [BJLR91, Corollary 15], we immediately get the following complexity upper bound for testing isomorphism for k-trees.

Corollary 18. *The isomorphism problem for k-trees is in* StUL*.*

4 *k*-Path Canonization

A k-path is a special type of k-tree. The subgraphs of k-paths are called partial k-paths. They coincide with the graphs of *pathwidth* at most k [Pro89]. In [GNPR05] a polynomial time algorithm for subgraph isomorphism for bounded pathwidth graphs was given. Here we look at the space complexity of the canonization problem for k-paths.

Definition 19. *An* interval graph *is a graph whose vertices can be put in one to one correspondence with a set of open intervals on the real line such that two vertices are adjacent if and only if the corresponding intervals have a nonempty intersection.*

Definition 20. [Klo94] *A* k-path *is a k-tree which is an interval graph.*

An alternative constructive definition of k-paths is given in [GNPR05]. The idea is to restrict the choice of the k-clique used as support for adding a new vertex depending on the support of the previously added vertex. The restriction can be best described by maintaining the notion of *current clique*.

Initially the starting clique is the current clique. When a new vertex is added it is joined to each vertex in the current clique. After adding the new vertex the current clique may remain the same (in that case the new vertex added becomes simplicial) or it may change by dropping a vertex and adding the new vertex in the current clique. Clearly, when a vertex is dropped it cannot come back in the current clique.

The difference between the definition of k-tree and the constructive definition of k-path is that for k-trees a new vertex can be joined to any k-clique when expanding a k-tree, whereas for k-paths a new vertex can only be added to the current clique of a k-path.

From this constructive definition of k-paths the following characterization of k-paths in the terminology of Section 3 can be obtained. Recall that a *caterpillar* is a rooted tree in which each node has at most one child that is not a leaf.

Lemma 21. *A k-tree G is a* k-path *if and only if for some base B of G, the tree $T(G,B)$ is a* caterpillar.

Proof (sketch). Assume that $G = (V, E)$ is a k-path and let C_i, $i = k, \ldots, n-1$, be the current k-clique that has been used as support for adding vertex v_{i+1} to $G_i = G[\{v_1, \ldots, v_i\}]$, where $C_k = \{v_1, \ldots, v_k\}$ is the initial k-clique. Notice that $C_i \neq C_{i+1}$ implies $C_j \neq C_i$ for all $j > i$. Now it is easy to verify that $T = T(G, C_1)$ is a caterpillar with vertices $C_k, (C_k, v_{k+1}), \ldots, (C_{n-1}, v_n)$ containing for each $j \geq k$ with $C_j = C_k$ the edge $(C_k, (C_k, v_{j+1}))$ and for each pair i, j with $C_i \neq C_{i+1} = C_j$ the edge $((C_i, v_{i+1}), (C_j, v_{j+1}))$.

For the backward direction assume that $T = T(G, B)$ is a caterpillar and let $B(0), \ldots, B(p)$ be the B-decomposition of G. We call $v \in V - B$ a leaf node if $(N_{i-1}(v), v)$ is a leaf in T. Now we can order the vertices of G in such a way that all the vertices in $B(i)$ precede the vertices in $B(i+1)$ and within each layer $B(i)$, $i > 0$, the leaf nodes come first. Let $v_1, \ldots, v_n$ be such an ordering. Then it is easy to verify that we can construct G from the initial k-tree $G_k = G[B] = G[\{v_1, \ldots, v_k\}]$ by successively adding the vertices v_{i+1} to $G_i = G[\{v_1, \ldots, v_i\}]$ using $N_i(v_{i+1})$ as the current clique. □

To canonize a given k-path G we use a similar approach as the one that we used in Section 3 for k-trees. In fact, the only difference is that now our algorithm A additionally checks for each base B whether $T(G, B)$ is a caterpillar. Notice that this can easily be done in logspace as follows.

Starting with the root B as the current node, the algorithm verifies that the current node has at most one child (C', v') in $T(G, B)$ that is not a leaf and then proceeds with (C', v') as the next current node (if the current node has two or more non leaf children, the algorithm detects that $T(G, B)$ is not a caterpillar).

As soon as the algorithm reaches a node that has only leaves as children it decides that $T(G, B)$ is a caterpillar and starts to compute the canons of the labeled trees $T(G, B, \theta)$ for all bijections $\theta : B \longrightarrow \{1, \ldots, k\}$ as explained in Section 3.

Since for a caterpillar $T(G, B)$ the oracle described in Claim 15 is clearly decidable in logspace, we have proved the following result.

Theorem 22. *For each fixed k there is a logspace canonizing algorithm for k-paths. Hence, the isomorphism problem for k-paths is in* L.

References

[AL89] Allender, E., Lange, K.-J.: RUSPACE(log n) is contained in DSPACE($\log^2$ n/loglog n). Theory of Computing Systems 31, 539–550 (1989)

[All04] Allender, E.: Arithmetic circuits and counting complexity classes. In: Krajíček, J. (ed.) Complexity of Computations and Proofs, Seconda Universita di Napoli. Quaderni di Matematica, vol. 13, pp. 33–72 (2004)

[AO96] Allender, E., Ogihara, M.: Relationships among PL, #L and the determinant. R.A.I.R.O. Informatique Théorique et Applications 30(1), 1–21 (1996)

[AP89] Arnborg, S., Proskurowski, A.: Linear time algorithms for NP-hard problems restricted to partial k-trees. Discrete Applied Mathematics 23(2), 11–24 (1989)

[Bab86] Babai, L.: A Las Vegas-NC algorithm for isomorphism of graphs with bounded multiplicity of eigenvalues. In: Proc. 27th IEEE Symposium on the Foundations of Computer Science, pp. 303–312. IEEE Computer Society Press, Los Alamitos (1986)

[BJLR91] Buntrock, G., Jenner, B., Lange, K.-J., Rossmanith, P.: Unambiguity and fewness for logarithmic space. In: Budach, L. (ed.) FCT 1991. LNCS, vol. 529, pp. 168–179. Springer, Heidelberg (1991)

[BL83] Babai, L., Luks, E.: Canonical labeling of graphs. In: Proc. 15th ACM Symposium on Theory of Computing, pp. 171–183 (1983)

[Bod88] Bodlaender, H.: Dynamic programming on graphs with bounded treewidth. In: Lepistö, T., Salomaa, A. (eds.) Automata, Languages and Programming. LNCS, vol. 317, pp. 105–118. Springer, Heidelberg (1988)

[Bod90] Bodlaender, H.: Polynomial algorithm for graph isomorphism and chromatic index on partial k-trees. Journal of Algorithms 11(4), 631–643 (1990)

[Bus97] Buss, S.: Alogtime algorithms for tree isomorphism, comparison, and canonization. In: Gottlob, G., Leitsch, A., Mundici, D. (eds.) KGC 1997. LNCS, vol. 1289, pp. 18–33. Springer, Heidelberg (1997)

[Cha90] Chandrasekharan, N.: Isomorphism testing of k-trees is in NC. Information Processing Letters 34(6), 283–287 (1990)

[CI88] Chandrasekharan, N., Iyengar, S.S.: NC algorithms for recognizing chordal graphs and k trees. IEEE Transactions on Computers 37(10), 1178–1183 (1988)

[Die97] Diestel, R.: Graph Theory. Graduate Texts in Mathematics, vol. 173. Springer, Heidelberg (1997)

[GNPR05] Gupta, A., Nishimura, N., Proskurowski, A., Ragde, P.: Embeddings of k-connected graphs of pathwidth k. Discrete Applied Mathematics 145(2), 242–265 (2005)

[GSS02] Del Greco, J.G., Sekharan, C.N., Sridhar, R.: Fast parallel reordering and isomorphism testing of k-trees. Algorithmica 32(1), 61–72 (2002)

[Gur97] Gurevich, Y.: From invariants to canonization. Bulletin of the European Association of Theoretical Computer Science (BEATCS) 63, 115–119 (1997)

[GV06] Grohe, M., Verbitsky, O.: Testing graph isomorphism in parallel by playing a game. In: Bugliesi, M., Preneel, B., Sassone, V., Wegener, I. (eds.) ICALP 2006. LNCS, vol. 4051, pp. 3–14. Springer, Heidelberg (2006)

[JKMT03] Jenner, B., Köbler, J., McKenzie, P., Torán, J.: Completeness results for graph isomorphism. Journal of Computer and System Sciences 66, 549–566 (2003)

[KCP82] Klawe, M.M., Corneil, D.G., Proskurowski, A.: Isomorphism testing in hookup classes. SIAM Journal of Algebraic Discrete Methods 3(2), 260–274 (1982)

[Klo94] Kloks, T. (ed.): Treewidth. LNCS, vol. 842. Springer, Heidelberg (1994)

[Lin92] Lindell, S.: A logspace algorithm for tree canonization. In: Proc. 24th ACM Symposium on Theory of Computing, pp. 400–404. ACM Press, New York (1992)

[Luk82] Luks, E.: Isomorphism of bounded valence can be tested in polynomial time. Journal of Computer and System Sciences 25, 42–65 (1982)

[Mil83] Miller, G.L.: Isomorphism of k-contractible graphs. Information and Computation 56(1/2), 1–20 (1983)

[Pro89] Proskurowski, A.: Maximal graphs of path-width k or searching a partial k-caterpillar. Technical Report UO-CIS-TR-89-17, University of Oregon (1989)

[Spi96] Spielman, D.A.: Faster isomorphism testing of strongly regular graphs. In: Proc. 28th ACM Symposium on Theory of Computing, pp. 576–584. ACM Press, New York (1996)

[SS87] Scheffler, P., Seese, D.: A combinatorial and logical approach to linear-time computability. In: Davenport, J.H. (ed.) ISSAC 1987 and EUROCAL 1987. LNCS, vol. 378, pp. 379–380. Springer, Heidelberg (1989)

[Wan94] Wanke, E.: Bounded tree-width and LOGCFL. Journal of Algorithms 16(3), 470–491 (1994)

Algorithms for Computing the Length-Constrained Max-Score Segments with Applications to DNA Copy Number Data Analysis

Hsiao-Fei Liu[1], Peng-An Chen[1], and Kun-Mao Chao[1,2,3,⋆]

[1] Department of Computer Science and Information Engineering
[2] Graduate Institute of Biomedical Electronics and Bioinformatics
[3] Graduate Institute of Networking and Multimedia
National Taiwan University, Taipei, Taiwan 106
kmchao@csie.ntu.edu.tw

Abstract. Given a sequence of n real numbers $A = (a_1, a_2, \ldots, a_n)$, two integers L and U with $1 \leq L \leq U \leq n$, and a score function $f : \mathbb{R}^+ \times \mathbb{R} \rightarrow \mathbb{R}$, the Length-Constrained Max-Score Segment Problem is to find a segment $A[i, j] = (a_i, a_{i+1}, \ldots, a_j)$ maximizing $f(j - i + 1, \sum_{h=i}^{j} a_h)$ subject to $j - i + 1 \in [L, U]$. In this paper, we solve the Length-Constrained Max-Score Segment Problem for the case where the given score function $f(\ell, w) = \frac{w}{\sqrt[r]{\ell}}$ for any constant $r > 1$. Our algorithm runs in $O(n\frac{T(L^{1/2})}{L^{1/2}})$ time, where $T(n')$ is the time required to solve the all-pairs shortest paths problem on a graph of n' nodes. By the latest result of Chan [7], $T(n') = O(n'^3 \frac{(\log\log n')^3}{(\log n')^2})$, so our algorithm runs in subquadratic time $O(nL\frac{(\log\log L)^3}{(\log L)^2})$. Lipson *et al.* [21] studied a more restricted case where the score function $f(\ell, w) = \frac{w}{\sqrt[2]{\ell}}$ and there are no length constraints, i.e., $L = 1$ and $U = n$. They also showed how to apply their algorithm to analyzing DNA copy number data. However, their algorithm takes $\Omega(n^2)$ time in the worst situation. Since the length lower bound L for the case considered by Lipson *et al.* is a constant, our algorithm solves it in $O(n)$ time.

1 Introduction

Given a sequence of n real numbers $A = (a_1, a_2, \ldots, a_n)$, two integers L and U with $1 \leq L \leq U \leq n$, and a score function $f : \mathbb{R}^+ \times \mathbb{R} \rightarrow \mathbb{R}$, define the length, weight, and score of a segment $A[i, j] = (a_i, \ldots, a_j)$ to be $\mathit{length}(i, j) = j - i + 1$, $\mathit{weight}(i, j) = \sum_{h=i}^{j} a_h$, and $\mathit{score}(i, j) = f(\mathit{length}(i, j), \mathit{weight}(i, j))$ respectively. The Length-Constrained Max-Score Segment Problem is to find a segment $A[i, j] = (a_i, a_{i+1}, \ldots, a_j)$ maximizing $\mathit{score}(i, j)$ subject to $\mathit{length}(i, j) \in [L, U]$.

⋆ Corresponding author.

T. Tokuyama (Ed.): ISAAC 2007, LNCS 4835, pp. 834–845, 2007.

1.1 Results

In this paper, we propose an algorithm to cope with the case where the score function $f(\ell, w) = \frac{w}{\sqrt[r]{\ell}}$ for a constant $r > 1$. Our algorithm runs in $O(n\frac{T(L^{1/2})}{L^{1/2}})$ time, where $T(n')$ is the time required to solve the all-pairs shortest paths problem on a graph of n' nodes. By the latest result of Chan [7], $T(n') = O(n'^3\frac{(\log\log n')^3}{(\log n')^2})$, so our algorithm runs in subquadratic time $O(nL\frac{(\log\log L)^3}{(\log L)^2})$. There was no known subquadratic time algorithm before even for the more restricted case: $f(\ell, w) = \frac{w}{\sqrt[2]{\ell}}$ and there are no length constraints, i.e., $L = 1$ and $U = n$. Our algorithm is the first subquadratic time algorithm which can cope with score functions of the form $f(\ell, w) = \frac{w}{\sqrt[r]{\ell}}$, where $r > 1$, and runs in linear time when there are no length constraints.

1.2 Related Work

The Length-Constrained Max-Score Segment Problem has been studied for the following cases.

- For the case where $f(\ell, w) = w$, Bernholt *et al.* [4], Fan *et al.* [10] , and Lin *et al.* [20] presented $O(n)$-time algorithms.
- For the case where $f(\ell, w) = w/\ell$, the problem is well studied in [4,8,13,16,18,20] and can be solved in $O(n)$ time using the algorithms proposed by Bernholt *et al.* [4], Chung and Lu [8], and Goldwasser *et al.* [13].
- For the case where f is quasiconvex, Bernholt *et al.* [4] proposed $O(n)$-time algorithms. A function $f : \mathbb{R}^+ \times \mathbb{R} \to \mathbb{R}$ is said to be *quasiconvex* if and only if for all points $u, v \in \mathbb{R}^+ \times \mathbb{R}$ and all $\lambda \in [0, 1]$, we have $f(\lambda \cdot u + (1 - \lambda) \cdot v) \leq \max\{f(u), f(v)\}$. Many known score functions, like w, $\frac{w}{\ell}$, and $\frac{|w|}{\sqrt[2]{\ell}}$, are quasiconvex; however, the score functions considered in this paper, like $\frac{w}{\sqrt[2]{\ell}}$ and $\frac{w}{\sqrt[3]{\ell}}$, are not quasiconvex.
- For the case where $f(\ell, w) = \frac{w}{\sqrt[2]{\ell}}$ and there are no length constraints, Lipson *et al.* [21] proposed an approximation scheme. Given any $\epsilon \in (0, 1/5]$, their approximation scheme runs in $O(n\epsilon^{-2})$ time and outputs a segment $A[i, j]$ such that $score(i, j)$ is at least $\mathrm{Opt}/\alpha(\epsilon)$, where Opt is the maximum segment score and $\alpha(\epsilon) = (1 - \sqrt{2\epsilon(2 + \epsilon)})^{-1}$. Note that our result immediately implies an $O(n)$-time exact algorithm for this case. To the best of our knowledge, it is the first exact algorithm with runtime better than $O(n^2)$ for this case.

1.3 Applications to DNA Copy Number Data Analysis

After the Human Genome Project was completed in 2003, many issues concerning human genome variations are raised [19,26]. Among all the different kinds of genome variations, copy number variations have attracted considerable attention and widely examined in the laboratory [17,28]. Copy number variations (CNVs)

of a target DNA sequence are DNA segments which are larger than 1kb and present at variable copy number in comparison with a reference genome. As we know, many genetic diseases, like cancer, are related to CNVs [25].

Let $P = (p_1, p_2, ..., p_n)$ be a sequence of short DNA segments (probes) sorted by their appearing locations in the DNA sequences. The approaches for measuring the DNA copy number changes, like array CGH [24], can provide a vector $V = (v_1, v_2, ..., v_n)$ of intensity signals such that if the target DNA sequence contains more copies of p_i than the reference DNA sequence does, the value of v_i will be higher than normal. On the other hand, if the target DNA sequence contains fewer copies of p_i than the reference DNA sequence does, the value of v_i will be lower than normal.

CNVs are caused by either amplification events (insertions or duplications of DNA segments) or deletion events (deletions of DNA segments) [12]. The goal is to locate where the events occurred. Specifically, we want to locate the amplification regions and the deletion regions. The amplification regions are regions $I_{amp} = [\ell_{amp}, r_{amp}] \subseteq [1, n]$ such that for all $i \in I_{amp}$, the target DNA sequence contains more copies of p_i than the reference DNA sequence does, so the intensity signal v_i with $i \in I_{amp}$ expects to have higher value than normal. The deletion regions are regions $I_{del} = [\ell_{del}, r_{del}] \subseteq [1, n]$ such that for all $i \in I_{del}$, the target DNA sequence contains fewer copies of p_i than the reference DNA sequence does, so the intensity signal v_i with $i \in I_{del}$ expects to have lower value than normal.

However, the intensity signals may be skewed due to noise. Thus, a statistical model for assessing the significance of amplification and deletion regions is needed. Lipson *et al*. [21] proposed a statistical model by assuming that the noise in the CNV data is independent for distinct probes. Denote by μ and σ the mean and standard derivation of the normal genomic data. Let the null hypothesis be that there are no events present in the target DNA sequence. Given a region I, define $\varphi^{sig}(I)$ by:

$$\varphi^{sig}(I) = \sum_{i \in I} \frac{v_i - \mu}{\sigma\sqrt{|I|}}.$$

When looking for amplification regions, we can let $s_{amp}(I) = \varphi^{sig}(I)$ be the score of region I. $s_{amp}(I)$ is then used to assess the significance of values in I. By the central limit theorem, the distribution of $\varphi^{sig}(I)$ is a normal distribution with mean = 0 and standard deviation = 1, and thus we have

$$Prob(s_{amp}(I) > z) \approx \frac{1}{2} \cdot \frac{1}{\sqrt{2\pi}} \cdot \frac{1}{z} e^{-\frac{1}{2}z^2}.$$

When looking for the deletion regions, we can let $s_{del}(I) = -\varphi^{sig}(I)$ be the score of region I. The distribution of $\varphi^{sig}(I)$ is a normal distribution with mean = 0 and standard deviation = 1, and thus we have

$$Prob(s_{del}(I) > z) \approx \frac{1}{2} \cdot \frac{1}{\sqrt{2\pi}} \cdot \frac{1}{z} e^{-\frac{1}{2}z^2}.$$

When we look for aberrant regions and don't care what kinds of the aberrance (amplification or deletion) are, we can let $s_A(I) = |\varphi^{sig}(I)|$ be the score of

region I. The distribution of $\varphi^{sig}(I)$ is a normal distribution with mean = 0 and standard deviation = 1, and thus we have

$$Prob(|\varphi^{sig}(I)| > z) \approx \frac{1}{\sqrt{2\pi}} \cdot \frac{1}{z} e^{-\frac{1}{2}z^2}.$$

Given a vector V, we then find the max-score region I_{max}, which is most possible to be the region we look for.

It is noticed that the central limit theorem can be applied reasonably only when the region I spans over more than than 25 probes [15]. Moreover, the lengths of amplification regions and deletion regions are usually less than 1Mb [26]. Thus we want to add length constraints on the located regions.

Bernholt *et al.*'s algorithm [4] can cope with the score function s_A in $O(n)$ time. Our algorithm can cope with the score functions s_{amp} and s_{del} in $O(n\frac{T(L^{1/2})}{L^{1/2}})$ time, where $T(n')$ is the time required to solve the all-pairs shortest paths problem on a graph with n' nodes. For the case where there are no length constraints, as considered by Lipson *et al.* [21], our algorithm runs in linear time.

2 Preliminaries

It is clear that $length(i,j) = j - i + 1$ can be evaluated in constant time. To evaluate $weight(i,j) = \sum_{h=i}^{j} a_h$ in constant time, we have to construct the prefix-sum array of A first. An array $\mathrm{PS}[0,\ldots,n]$ is said to be the prefix-sum array of A if and only if $\mathrm{PS}[i] = a_1 + a_2 + \ldots + a_i$ for each $i > 0$ and $\mathrm{PS}[0] = 0$. The prefix-sum array $\mathrm{PS}[0,\ldots,n]$ can be computed in linear time by setting $\mathrm{PS}[0]$ to 0 and $\mathrm{PS}[i]$ to $\mathrm{PS}[i-1] + a_i$ for i from 1 to n. After constructing the prefix-sum array $\mathrm{PS}[0,\ldots,n]$, each evaluation of $weight(i,j)$ can be done in constant time because $weight(i,j) = \mathrm{PS}[j] - \mathrm{PS}[i-1]$. When analyzing the running time, we will assume that the evaluation of $f(\ell, w)$ can be done in constant time. It is obviously the case for the typically-considered functions like $f(\ell, w) = w$ and $f(\ell, w) = w/\sqrt{\ell}$.

In the following, we review some definitions and theorems. For more details, readers can refer to [1,4,5,10,30].

Definition 1. *A score function $f : \mathbb{R}^+ \times \mathbb{R} \to \mathbb{R}$ is said to be* quasiconvex *if and only if for all points $u, v \in \mathbb{R}^+ \times \mathbb{R}$ and all $\lambda \in [0,1]$, we have $f(\lambda \cdot u + (1-\lambda) \cdot v) \leq \max\{f(u), f(v)\}$.*

Lemma 1. [5] *Let $r > 1$. Define $f' : \mathbb{R}^+ \times \mathbb{R}$ by letting*

$$f'(\ell, w) = \begin{cases} \frac{w}{\sqrt[r]{\ell}} & \text{if } w \geq 0; \\ 0 & \text{otherwise.} \end{cases}$$

Then f' is quasiconvex.

Theorem 1. [4] *Given a sequence of n real numbers $A = (a_1, a_2, \ldots, a_n)$, two integers L and U with $1 \leq L \leq U \leq n$, and a score function $f : \mathbb{R}^+ \times \mathbb{R} \to \mathbb{R}$,*

there exists an algorithm, denoted by MSSQ(A, L, U, f), which can solve the Length-Constrained Max-Score Segment Problem *in linear time if the given score function f is quasiconvex.*

By the fact that $f(\ell, w) = w$ is quasiconvex and Theorem 1, we have the following corollary, which was also proved in [10,20].

Corollary 1. *There exists an $O(n)$-time algorithm for finding a segment $A[p', q']$ maximizing $weight(p', q')$ subject to $length(p', q') \in [L, U]$.*

By Corollary 1, we can find the max-weight segment $A[p', q']$ in linear time. Clearly, if $weight(p', q') = 0$, then $A[p', q']$ is also the max-score segment. If $weight(p', q') > 0$, then we know there is at least one segment satisfying the length constraints with positive weight. Thus we can first change the original score function $f(\ell, w) = \frac{w}{\sqrt[r]{\ell}}$ to

$$f'(\ell, w) = \begin{cases} \frac{w}{\sqrt[r]{\ell}} \text{ if } w \geq 0; \\ 0 \quad \text{otherwise,} \end{cases}$$

and then call MSSQ(A, L, U, f') to find the max-score segment in linear time. However, when $weight(p', q') < 0$, i.e., all segments satisfying the length constraints have negative weights, the situation becomes complex. The following lemma is useful when we have to face this complex situation.

Lemma 2. *Let $r > 1$ and score function $f(\ell, w) = \frac{w}{\sqrt[r]{\ell}}$. If $weight(p, q) < 0$ for all (p, q) with $length(p, q) \in [L, U]$, then $length(p^*, q^*) < 2L$, where $(p^*, q^*) = \arg \max_{length(p,q) \in [L,U]} score(p, q)$.*

Proof. Let $(p^*, q^*) = \arg \max_{length(p,q) \in [L,U]} score(p, q)$. Suppose for the contradiction that $lenght(p^*, q^*) \geq 2L$. Let $c_1 = \lfloor (p^* + q^*)/2 \rfloor$ and $c_2 = c_1 + 1$. Then we have $length(p^*, c_1) \in [L, U]$ and $length(c_2, q^*) \in [L, U]$. Since $\frac{weight(p^*,q^*)}{length(p^*,q^*)} = \frac{weight(p^*,c_1)+weight(c_2,q^*)}{length(p^*,c_1)+weight(c_2,q^*)}$, we have

$$\frac{weight(p^*, q^*)}{length(p^*, q^*)} \leq \frac{weight(p^*, c_1)}{length(p^*, c_1)} \text{ or } \frac{weight(p^*, q^*)}{length(p^*, q^*)} \leq \frac{weight(c_2, q^*)}{weight(c_2, q^*)}.$$

Without loss of generality, we assume $\frac{weight(p^*,q^*)}{length(p^*,q^*)} \leq \frac{weight(p^*,c_1)}{length(p^*,c_1)}$. Since $\frac{weight(p^*,q^*)}{length(p^*,q^*)} \leq \frac{weight(p^*,c_1)}{length(p^*,c_1)} < 0$ and $length(p^*, q^*)^{1-1/r} > length(p^*, c_1)^{1-1/r}$, we have

$$\begin{aligned} & \frac{length(p^*, q^*)^{1-1/r} \cdot weight(p^*, q^*)}{length(p^*, q^*)} < \frac{length(p^*, c_1)^{1-1/r} \cdot weight(p^*, c_1)}{length(p^*, c_1)} \\ \Leftrightarrow & \frac{weight(p^*, q^*)}{length(p^*, q^*)^{1/r}} < \frac{weight(p^*, c_1)}{length(p^*, c_1)^{1/r}} \\ \Leftrightarrow & score(p*, q*) < score(p^*, c_1). \end{aligned}$$

It contradicts that $(p^*, q^*) = \arg \max_{length(p,q) \in [L,U]} score(p, q)$. □

3 Min-Plus Convolution

In the following we shall design an algorithm for the MIN-PLUS CONVOLUTION PROBLEM. Although it appears to be a digression at first, we shall reveal its relevance to our original problem in the next section (see Lemma 5). The min-plus convolution of two vectors $\mathbf{x} = (x_0, x_1, \ldots, x_{n-1})$ and $\mathbf{y} = (y_0, y_1, \ldots, y_{n-1})$ is a vector $\mathbf{z} = (z_0, z_1, \ldots, z_{n-1})$ such that $z_k = \min_{i=0}^{k}\{x_i + y_{k-i}\}$ for $k = 0, 1, \ldots, n-1$. Given two vectors $\mathbf{x} = (x_0, x_1, \ldots, x_{n-1})$ and $\mathbf{y} = (y_0, y_1, \ldots, y_{n-1})$, the MIN-PLUS CONVOLUTION PROBLEM is to compute the min-plus convolution $\mathbf{z}$ of $\mathbf{x}$ and $\mathbf{y}$. This problem has appeared in the literature with various names such as "minimum convolution," "epigraphical sum," "inf-convolution," and "lowest midpoint" [2,3,11,22,23,27,29]. Although it is easy to obtain an $O(n^2)$-time algorithm, no subquadratic algorithm was known until recently Bremner *et al.* [6] proposed an $O(n^2/\log n)$-time algorithm. In the following, we shall give an $O(n^{1/2}T(n^{1/2}))$-time algorithm for the MIN-PLUS CONVOLUTION PROBLEM, where $T(n)$ is the time required to solve the all-pairs shortest paths problem on a graph of n nodes. To date, the best algorithm for computing the all-pairs shortest paths problem on a graph of n nodes runs in $O(n^3\frac{(\log\log n)^3}{(\log n)^2})$ time [7]. Thus, our work implies an $O(n^2\frac{(\log\log n)^3}{(\log n)^2})$-time algorithm for the min-plus convolution problem, which is slightly superior to the first subquadratic $O(n^2/\log n)$-time algorithm recently proposed by Bremner *et al.* [6].

Definition 2. *The min-plus product BC of a $d \times n'$ matrix $B = [b_{i,j}]$ and an $n' \times d$ matrix $C = [c_{i,j}]$ is a $d \times d$ matrix $D = [d_{i,j}]$ where $d_{i,j} = \min_{k=0}^{n'-1}\{b_{i,k} + c_{k,j}\}$.*

Note that the notion of "min-plus product" is different from the notion of "min-plus convolution". It is well known [1] that the time complexity of computing the min-plus product of two $n' \times n'$ matrices is asymptotically equal to that of computing all pairs shortest paths for a graph with n' vertices. The proof of the next lemma was also given in [30], and we include it here for completeness.

Lemma 3. *Given a $T(n')$-time algorithm for computing the min-plus product of any two $n' \times n'$ matrices, the computation of the min-plus product of B and C, where B is a $d \times n'$ matrix and C is an $n' \times d$ matrix, can be done in $O(\frac{n'}{d}T(d))$ time if $d \leq n'$.*

Proof. For simplicity we assume that d divides n. We first split B into n'/d matrices $B_1, \ldots, B_{n'/d}$ of dimension $d \times d$ and C into n'/d matrices $C_1, \ldots, C_{n'/d}$ of dimension $d \times d$. Then we can compute $\{B_1C_1, B_2C_2, \ldots, B_{n'/d}C_{n'/d}\}$ in $O(dT(n'/d))$ time by the given algorithm. The (i,j)-th entry of the min-plus product of B and C is $\min_{k=1}^{n'/d}\{\text{the } (i,j)\text{-th entry of } B_kC_k\}$. □

Our new algorithm for the MIN-PLUS CONVOLUTION PROBLEM is as follows.

Algorithm: MinPlusConvolution
Input: $\mathbf{x} = (x_0, x_1, \ldots, x_{n-1})$ and $\mathbf{y} = (y_0, y_1, \ldots, y_{n-1})$.
Output: $\mathbf{z} = (z_0, z_1, \ldots, z_{n-1})$ such that $z_k = \min_{i=0}^{k}\{x_i + y_{k-i}\}$ for $k = 0, 1, \ldots, n-1$.

1. Construct an $\lceil n^{1/2} \rceil \times (2n-1)$ matrix $B = [b_{i,j}]$ such that the i^{th} row of B is equal to $(\infty, \ldots, \infty, x_0, x_1, \ldots, x_{n-1}, \underbrace{\infty, \ldots, \infty}_{i \times \lceil n^{1/2} \rceil})$ for $i = 0, 1, \ldots, \lceil n^{1/2} \rceil - 1$.
2. Construct a $(2n-1) \times \lceil n^{1/2} \rceil$ matrix $C = [c_{i,j}]$ such that the transpose of the j^{th} column of C is equal to $(\underbrace{\infty, \ldots, \infty}_{j}, y_{n-1}, y_{n-2}, \ldots, y_0, \infty, \ldots, \infty)$ for $j = 0, 1, \ldots, \lceil n^{1/2} \rceil - 1$.
3. Let $D = [d_{i,j}]$ be the min-plus product of B and C.
4. For $k = 0, 1, \ldots, n-1$ do
   Find i, j such that $k = i \times \lceil n^{1/2} \rceil + j$, where $0 \le j < \lceil n^{1/2} \rceil$.
   Set z_k to $d_{i,j}$.
5. Output $\mathbf{z} = (z_0, z_1, \ldots, z_{n-1})$.

The following lemma ensures the correctness.

Lemma 4. *In* MinPlusConvolution, $d_{i,j} = \min_{t=0}^{i \times \lceil n^{1/2} \rceil + j}\{x_t + y_{i \times \lceil n^{1/2} \rceil + j - t}\}$ *if* $0 \le i \times \lceil n^{1/2} \rceil + j \le n-1$.

Proof.

$$
\begin{aligned}
d_{i,j} &= \min_{t=0}^{2n-1}\{b_{i,t} + c_{t,j}\} \\
&= \min\{\min_{t=0}^{n-2-i\times\lceil n^{1/2}\rceil}\{b_{i,t} + c_{t,j}\}, \min_{t=n-1-i\times\lceil n^{1/2}\rceil}^{n+j-1}\{b_{i,t} + c_{t,j}\}, \min_{t=n+j}^{2n-1}\{b_{i,t} + c_{t,j}\}\} \\
&= \min\{\min_{t=0}^{n-2-i\times\lceil n^{1/2}\rceil}\{\infty + c_{t,j}\}, \min_{t=n-1-i\times\lceil n^{1/2}\rceil}^{n+j-1}\{b_{i,t} + c_{t,j}\}, \min_{t=n+j}^{2n-1}\{b_{i,t} + \infty\}\} \\
&= \min_{t=n-1-i\times\lceil n^{1/2}\rceil}^{n+j-1}\{b_{i,t} + c_{t,j}\} \\
&= \min\{x_0 + y_{i\times\lceil n^{1/2}\rceil+j}, x_1 + y_{i\times\lceil n^{1/2}\rceil+j-1}, \ldots, x_{i\times\lceil n^{1/2}\rceil+j} + y_0\} \\
&= \min_{t=0}^{i\times\lceil n^{1/2}\rceil+j}\{x_t + y_{i\times\lceil n^{1/2}\rceil+j-t}\}
\end{aligned}
$$

□

We now analyze the time complexity. Let $T(n)$ denote the time required to compute the min-plus product of two $n \times n$ matrices. Steps 1 and 2 take $O(n^{3/2})$ time, and by Lemma 3, Step 3 takes $O(\frac{2n-1}{n^{1/2}} T(\lceil n^{1/2} \rceil)) = O(n^{1/2} T(n^{1/2}))$ time.

Steps 4 and 5 take $O(n)$ time. Therefore, the total runtime is $O(n^{1/2}T(n^{1/2}) + n^{3/2})$. Since $T(n) = \Omega(n^2)$, we have $O(n^{1/2}T(n^{1/2}) + n^{3/2}) = O(n^{1/2}T(n^{1/2}))$. Theorem 2 summarizes our work for the MIN-PLUS CONVOLUTION PROBLEM.

Theorem 2. *The running time of* MINPLUSCONVOLUTION *is* $O(n^{1/2}T(n^{1/2}))$, *where* $T(n)$ *is the time required to compute the min-plus product of two* $n \times n$ *matrices.*

4 The Algorithm

In this section, we show how to solve the LENGTH-CONSTRAINED MAX-SCORE SEGMENT PROBLEM in $O(n\frac{T(L^{1/2})}{L^{1/2}})$ time for the case where the given score function $f(\ell, w) = \frac{w}{\sqrt[r]{\ell}}$ for a constant $r > 1$. To make use of the min-plus convolution algorithm developed in the previous section, we need the following Lemma.

Lemma 5. [3] *Given a sequence* $X = (x_1, \ldots, x_n)$, *the* MAXIMUM CONSECUTIVE WEIGHTS PROBLEM *is to compute a sequence* $W = (w_1, \ldots, w_n)$ *where* $w_i = \max\{x_p + x_{p+1} + \ldots + x_q | q - p + 1 = i\}$ *for each* $i = 1, ..., n$. *The* MAXIMUM CONSECUTIVE WEIGHTS PROBLEM *can be reduced to the* MIN-PLUS CONVOLUTION PROBLEM *in linear time.*

The next corollary follows directly from Theorem 2 and Lemma 5.

Corollary 2. *Given a sequence* $X = (x_1, \ldots, x_n)$, *we can compute a sequence* $W = (w_1, \ldots, w_n)$ *in which* $w_i = \max\{x_p + x_{p+1} + \ldots + x_q | q - p + 1 = i\}$ *for each* $i = 1, ..., n$ *in* $O(n^{1/2}T(n^{1/2}))$ *time, where* $T(n)$ *is the time required to compute the min-plus product of two* $n \times n$ *matrices.*

We now describe our algorithm. To avoid notational overload, we assume that $4L$ divides n. First we have to compute $(p', q') = \arg \max_{length(p',q') \in [L,U]} weight(p', q')$. Then there are three cases to consider: (1) $weight(p', q') = 0$; (2) $weight(p', q') > 0$; (3) $weight(p', q') < 0$. If it is Case 1, we know $A[p', q']$ is already a correct solution. If it is Case 2, then we know there is at least one segment satisfying the length constraints with positive weight. By Lemma 1 and Theorem 1, we can find the length-constrained max-score segment in linear time. If it is Case 3, by letting A_i be $A[2iL+1, 2iL+4L], i = 0, 1, 2, ..., \frac{n}{2L} - 2$, we can divide the sequence A into $\frac{n}{2L} - 1$ segments of length $4L$. See Figure 1 for illustration. By making use of Corollary 2, we are able to compute the length-constrained max-score segment s_i^* contained in A_i in $O(L^{1/2}T(L^{1/2}))$ time for each $i = 0, 1, ..., \frac{n}{2L} - 2$. By Lemma 2, some s_i^* must be the correct solution. The detailed algorithm is given below.

Algorithm MSS(A, L, U, f)
Input: a sequence $A = (a_1, a_2, \ldots, a_n)$, two integers L and U with $1 \leq L \leq U \leq n$, and score function $f : \mathbb{R}^+ \times \mathbb{R} \to \mathbb{R}$ with $f(\ell, w) = \frac{w}{\sqrt[r]{\ell}}$ for any constant $r > 1$.

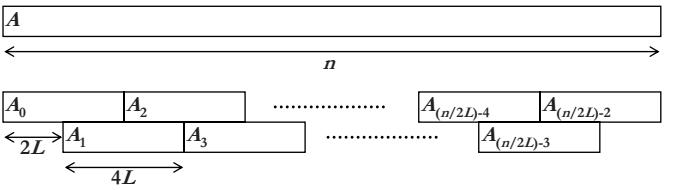


Fig. 1. Illustration of $A_i, i = 0, 1, 2, ..., \frac{n}{2L} - 2$

Output: a segment $A[p, q]$ maximizing $score(p, q)$ subject to $L \leq length(p, q) \leq U$.

1. Compute $(p', q') = \arg \max_{length(p',q') \in [L,U]} weight(p', q')$.
2. If $weight(p', q') = 0$, then return $A[p', q']$.
3. If $weight(p', q') > 0$ then
   1. define $f' : \mathbb{R}^+ \times \mathbb{R}$ by letting $f'(\ell, w) = \frac{w}{\sqrt[r]{\ell}}$ if $w \geq 0$ and 0, otherwise;
   2. return MSSQ$(A[1, n], L, U, f')$.
4. Else, execute the following steps.
   1. For i from 0 to $\frac{n}{2L} - 2$,
      (a) compute $(w_1, \ldots, w_{4L})$, where $w_j = \max\{weitht(p, q) | length(p, q) = j$ and $2iL+1 \leq p \leq q \leq 2iL+4L\}$ for each $i = 1, \ldots, 4L$;
      (b) compute $s_j = \frac{w_j}{\sqrt[r]{j}}$ for each $j = 1, \ldots, 4L$;
      (c) let $s_{i^*} = \max\{s_L, \ldots, s_{\min\{2L-1,U\}}\}$;
      (d) set s_i^* to the max-score segment in $\{A[p, q] | length(p, q) = i^*$ and $2iL + 1 \leq p \leq q \leq 2iL + 4L\}$.
   2. Return the max-score segment in $\{s_0^*, s_1^*, \ldots, s_{\frac{n}{2L}-2}^*\}$.

Theorem 3. *Given a sequence $A = (a_1, a_2, \ldots, a_n)$, two integers L and U with $1 \leq L \leq U \leq n$, and score function $f : \mathbb{R}^+ \times \mathbb{R} \to \mathbb{R}$ with $f(\ell, w) = \frac{w}{\sqrt[r]{\ell}}$, MSS$(A, L, U, f)$ finds a segment $A[p, q]$ maximizing $f(j-i+1, \sum_{h=i}^{j} a_h)$ subject to $j - i + 1 \in [L, U]$ in $O(n\frac{T(L^{1/2})}{L^{1/2}})$ time, where $T(n')$ is the time required to compute the min-plus product of two $n' \times n'$ matrices.*

Proof. We begin by considering the correctness. Let $(p', q') = \arg \max_{length(p',q') \in [L,U]} weight(p', q')$ and $(p^*, q^*) = \arg \max_{length(p^*,q^*) \in [L,U]} score(p^*, q^*)$.

In the case where $weight(p', q') = 0$, we have $score(p, q) = \frac{weight(p,q)}{\sqrt[r]{length(p,q)}} \leq 0$ for all (p, q) with $length(p, q) \in [L, U]$. It follows that $0 \geq score(p^*, q^*) \geq score(p', q') = 0$, so $A[p', q']$ is a correct solution.

In the case where $weight(p', q') > 0$, we have $score(p^*, q^*) \geq score(p', q') > 0$. Let $f'(\ell, w) = \frac{w}{\sqrt[r]{\ell}}$ if $w \geq 0$ and 0, otherwise. Since $score(p^*, q^*) > 0$, it does not matter to replace f with f', i.e., doing so won't change the solution. Furthermore, f' is quasiconvex by Lemma 1. Thus by Theorem 1, MSSQ(A, L, U, f') will return a correct solution.

In the case where $weight(p', q') < 0$, we have $length(p^*, q^*) < 2L$ by Lemma 2. It follows that $A[p^*, q^*]$ must be contained in $A[2iL + 1, 2iL + 4L]$ for some $i \in [0, \frac{n}{2L} - 2]$ and thus the solution returned at Step 4 must be correct.

Now we analyze the runtime. By Corollary 1, Step 1 takes $O(n)$ time. Step 2 takes constant time. By Theorem 1, Step 3 takes $O(n)$ time. By Corollary 2, each iteration of the loop at Step 4.1 takes $O(L^{1/2}T(L^{1/2}) + L)$ time. Thus Step 4.1 takes $O(\frac{n}{L}L^{1/2}T(L^{1/2}) + n) = O(n\frac{T(L^{1/2})}{L^{1/2}})$ time. Step 4.2 takes $O(n/L)$ time. Therefore the total runtime is $O(n\frac{T(L^{1/2})}{L^{1/2}})$. □

5 Concluding Remarks

In this paper, we study the problem of finding the max-score segment of a sequence with length constraints. We give an $O(nL\frac{(\log\log L)^3}{(\log L)^2})$-time algorithm to cope with any score function of the form $f(\ell, w) = \frac{w}{\sqrt[r]{\ell}}$, where $r \geq 1$. Even for the case where there are no length constraints, there was no known subquadratic time algorithm for this class of score functions before. Our algorithm is the first subquadratic time algorithm for this class of score functions and runs in linear time when there are no length constraints. To our best knowledge, there is not any non-trivial lower bound proved so far. Thus, there is still a large gap between the trivial lower bound of $O(n)$ and the upper bound of $O(nL\frac{(\log\log L)^3}{(\log L)^2})$. Bridging this gap remains an open problem.

Acknowledgments

We thank the anonymous referees for helpful suggestions. We thank Kuan-Yu Chen, Meng-Han Li, Cheng-Wei Luo, Hung-Lung Wang, and Roger Yang for helpful discussion.

References

1. Aho, A., Hopcroft, J., Ullman, J.: The Design and Analysis of Computer Algorithms. Addison-Wesley, Reading (1974)
2. Bellman, R., Karush, W.: Mathematical Programming and the Maximum Transform. Journal of the Society for Industrial and Applied Mathematics 10(3), 550–567 (1962)
3. Bergkvist, A., Damaschke, P.: Fast Algorithms for Finding Disjoint Subsequences with Extremal Densities. Pattern Recognition 39(12), 2281–2292 (2006)
4. Bernholt, T., Eisenbrand, F., Hofmeister, T.: A Geometric Framework for Solving Subsequence Problems in Computational Biology Efficiently. In: SoCG, pp. 310–318 (2007)
5. Bernholt, T., Hofmeister, T.: An Algorithm for a Generalized Maximum Subsequence Problem. In: Correa, J.R., Hevia, A., Kiwi, M. (eds.) LATIN 2006. LNCS, vol. 3887, pp. 178–189. Springer, Heidelberg (2006)

6. Bremner, D., Chan, T., Demaine, E., Erickson, J., Hurtado, F., Iacono, J., Langerman, S., Streinu, I., Taslakian, P.: Necklaces, Convolutions, and X+Y. In: Azar, Y., Erlebach, T. (eds.) ESA 2006. LNCS, vol. 4168, pp. 160–171. Springer, Heidelberg (2006)
7. Chan, T.M.: More Algorithms for All-Pairs Shortest Paths in Weighted Graphs. In: STOC (to appear, 2007)
8. Chung, K.-M., Lu, H.-I.: An Optimal Algorithm for the Maximum-Density Segment Problem. SIAM Journal on Computing 34(2), 373–387 (2004)
9. de Berg, M., van Kreveld, M., Overmars, M., Schwarzkopf, O.: Computational Geometry: Algorithms and Applications, 2nd edn. Springer, Heidelberg (2000)
10. Fan, T.-H., Lee, S., Lu, H.-I., Tsou, T.-S., Wang, T.-C., Yao, A.: An Optimal Algorithm for Maximum-Sum Segment and Its Application in Bioinformatics Extended Abstract. In: Ibarra, O.H., Dang, Z. (eds.) CIAA 2003. LNCS, vol. 2759, pp. 251–257. Springer, Heidelberg (2003)
11. Felzenszwalb, P., Huttenlocher, D.: Distance Transforms of Sampled Functions. Technical Report TR2004-1963, Cornell Computing and Information Science (2004)
12. Feuk, L., Carson, A.R., Scherer, S.W.: Structural variation in the human genome. Nature Reviews Genetics 7, 85–97 (2006)
13. Goldwasser, M., Kao, M.-Y., Lu, H.-I.: Linear-Time Algorithms for Computing Maximum-Density Sequence Segments with Bioinformatics Applications. Journal of Computer and System Sciences 70(2), 128–144 (2005)
14. Han, Y.: An $O(n^3(\log\log n/\log n)^{5/4})$ Time Algorithm for All Pairs Shortest Path. In: Azar, Y., Erlebach, T. (eds.) ESA 2006. LNCS, vol. 4168, pp. 411–417. Springer, Heidelberg (2006)
15. Hogg, R.V., Tanis, E.A.: Probability and Statistical Inference, 7th edn. (2005)
16. Huang, X.: An Algorithm for Identifying Regions of a DNA Sequence that Satisfy a Content Requirement. Computer Applications in the Biosciences 10(3), 219–225 (1994)
17. Iafrate, A J., Feuk, L., Rivera, M.N., Listewnik, M.L, Donahoe, P.K, Qi, Y., Scherer, S.W, Lee, C.: Detection of Large-Scale Variation in the Human Genome. Nature Genetics 36(9), 949–951 (2004)
18. Kim, S.K.: Linear-Time Algorithm for Finding a Maximum-Density Segment of a Sequence. Information Processing Letters 86(6), 339–342 (2003)
19. Komura, D., Shen, F., Ishikawa, S., Fitch, K.R., Chen, W., Zhang, J., Liu, G., Ihara, S., Nakamura, H., Hurles, M.E., et al.: Genome-wide Detection of Human Copy Number Variations Using High-Density DNA Oligonucleotide Arrays. Genome Research 16(12), 1575–1584 (2006)
20. Lin, Y.-L., Jiang, T., Chao, K.-M.: Efficient Algorithms for Locating the Length-Constrained Heaviest Segments with Applications to Biomolecular Sequence Analysis. Journal of Computer and System Sciences 65(3), 570–586 (2002)
21. Lipson, D., Aumann, Y., Ben-Dor, A., Linial, N., Yakhini, Z.: Efficient Calculation of Interval Scores for DNA Copy Number Data Analysis. In: McLysaght, A., Huson, D.H. (eds.) RECOMB 2005. LNCS (LNBI), vol. 3678, pp. 83–100. Springer, Heidelberg (2005)
22. Maragos, P.: Differential Morphology. Nonlinear Image Processing, 289–329 (2000)
23. Moreau, J.-J.: Inf-Convolution, Sous-Additivité, Convexité Des Fonctions Numériques. Journal de Mathématiques Pures et Appliquées 49, 109–154 (1970)

24. Pinkel, D., Segraves, R., Sudar, D., Clark, S., Poole, I., Kowbel, D., Collins, C., Kuo, W.-L., Chen, C., Zhai, Y., et al.: High Resolution Analysis of DNA Copy Number Variation Using Comparative Genomic Hybridization to Microarrays. Nature Genetics 20(2), 207–211 (1998)
25. Pollack, J.R., Perou, C.M., Alizadeh, A.A., Eisen, M.B., Pergamenschikov, A., Williams, C.F., Jeffrey, S.S., Botstein, D., Brown, P.O.: Genome-wide analysis of DNA copy-number changes using cDNA microarrays. Nature Genetics 23(1), 41–46 (1999)
26. Redon, R., Ishikawa, S., Fitch, K.R., Feuk, L., Perry, G.H., Andrews, T.D., Fiegler, H., Shapero, M.H., Carson, A.R., Chen, W.: Global Variation in Copy Number in the Human Genome. Nature 444, 444–454 (2006)
27. Rockafellar, R.T.: Convex Analysis (1970)
28. Sebat, J., Lakshmi, B., Troge, J., Alexander, J., Young, J., Lundin, P., Måner, S., Massa, H., Walker, M., Chi, M., et al.: Large-Scale Copy Number Polymorphism in the Human Genome. Science 305(23), 525–528 (2004)
29. Strömberg, T.: The Operation of Infimal Convolution. Dissertationes Mathematicae 352, 58 (1996)
30. Takaoka, T.: Efficient Algorithms for the Maximum Subarray Problem by Distance Matrix Multiplication. Electronic Notes in Theoretical Computer Science 61, 191–200 (2002)

Space Efficient Indexes for String Matching with Don't Cares

Tak-Wah Lam[1,⋆], Wing-Kin Sung[2], Siu-Lung Tam[1], and Siu-Ming Yiu[1]

[1] Department of Computer Science, University of Hong Kong
{twlam,sltam,smyiu}@cs.hku.hk
[2] Department of Computer Science, National University of Singapore
ksung@comp.nus.edu.sg

Abstract. Given a text T of length n, the classical indexing problem for pattern matching is to build an index for T so that for any query pattern P, we can report efficiently all occurrences of P in T. Cole et al (2004) extended this problem to allow don't care characters (wildcards) in the text and pattern, and they gave the first index that supports efficient pattern matching. The space complexity of this index is linear in n (text length) but exponential in terms of the number of wildcards. Motivated by bioinformatics applications, we investigate indexes whose size depends on n only. In the literature, space efficient indexes for wildcard matching are known only for the special case when wildcards appear only in the pattern (Iliopoulos and Rahman, 2007); the space required is $O(n)$. Not much has been heard for the case when wildcards appear in the text only, or in both the text and pattern. In this paper we give an $O(n)$ space index to support efficient wildcard matching in all three cases. Our solution to the pattern-only case improves the matching time of the previous work tremendously in practice. In addition, our solution can be extended to handle optional wildcards, each of which can match zero or one character.

1 Introduction

The classical indexing problem for pattern matching is to build an index for a text T so that for any query pattern P we can report efficiently all positions s such that $T[s\,..\,s+|P|-1] = P$. We call these positions s the matches of P in T. The problem is fundamental and finds application in many areas such as text retrieval, computational biology, data mining and network security. Near optimal solutions were known since 1970s. Suffix trees [9,7] use $O(n)$ space and achieve the optimal searching time, i.e. $O(m + occ)$, where n and m are the length of T and P, respectively, and occ is the number of matches of P in T.

Some applications however require more sophisticated forms of searching. Fischer and Paterson [4] extended the pattern matching problem by introducing wildcards. A wildcard, denoted as ϕ, is a special character that can match any character. In their problem setting, we are given P and T, both of which

⋆ Part of the work is supported by Hong Kong RGC Grant 7140/06E.

T. Tokuyama (Ed.): ISAAC 2007, LNCS 4835, pp. 846–857, 2007.

may contain wildcards, and we need to find all matches of P in T. Cole et al. [3] further considered indexing the text to speed up the matching process. Given a text with $k \geq 0$ wildcards and an integer $d \geq 0$, they showed how to build an index of $O(n \log^{k+d} n)$ space to allow searching for any pattern with at most d wildcards. For a pattern containing $g \leq d$ wildcards, the matching takes $O(m + 2^g \log^k n \log\log n + occ)$ time[1]. The drawback of this solution is its enormous space requirement. In bioinformatics applications, the text is typically a genome with millions or even billions of characters long. The factor $\log^{k+d} n$ would imply a prohibitive amount of memory for even a few wildcard characters in either the text or pattern. Another drawback is that the index, once built for a fixed d, only allows searching for patterns with at most d wildcards.

A recent work [6] by Iliopoulos and Rahman gave an $O(n)$ space index for a text without wildcard to support searching for a pattern with any number of wildcards. To understand the time complexity, we need the following notations. Denote a pattern $P = P_1\phi^{g_1}P_2\phi^{g_2}\cdots\phi^{g_h}P_{h+1}$ where ϕ^{g_i} denotes a group of $g_i \geq 1$ wildcards, and each P_i is a *pattern segment* containing no wildcards. Let α be the sum of the number of matches of every pattern segment P_i in T. They gave an index for T with $O(n)$ space and support searching for all matches of P in T in $O(m + \alpha)$ time. The drawback is that one malicious pattern segment (say, a very short one) may contribute many matches and make α really big. Searching would be slow since α maybe $\Theta(n)$. To the best of our knowledge, space efficient solution for indexing text with wildcards has not been known.

Following the previous works, we naturally consider three different settings: wildcards can be present in the pattern only, the text only, and both the text and the pattern. We give $O(n)$ space indexes for all three settings. In particular, our index for wildcards in pattern only improves the searching time of Iliopoulos and Rahman [6]. Analogous to partitioning P into pattern segments, we denote $T = T_1\phi^{k_1}T_2\phi^{k_2}\cdots\phi^{k_\ell}T_{\ell+1}$ for a text with ℓ groups of wildcards. For ease of discussion, we use $occ(X, Y)$ to denote the total number of matches of string X in string Y, considering wildcards in the strings if any. Note that $occ = occ(P, T)$.

A. Wildcards in pattern only. Given a text T without wildcard. We can build an $O(n)$ space index so that, for any pattern P with h groups of wildcards, searching takes $O(m+h\beta)$ time, where $\beta = \min_{1\leq i\leq h+1} occ(P_i, T)$. Note that $h\beta \leq \alpha$ (recall that $\alpha = \sum_{i\leq i\leq h+1} occ(P_i, T)$), and the searching time of our index is upper bounded by that of Iliopoulos and Rahman [6]. Intuitively, β is large only when all P_i generate plenty of matches. The case where $\beta = \Theta(n)$ still exists, but is less likely to happen.

B. Wildcards in text only. Given a text T with ℓ groups of wildcards. We can build an $O(n)$ space index so that, for any pattern P without wildcard, searching P takes $O(m \log n + \gamma + occ)$ time, where $\gamma = \sum_{j=1}^{\ell+1} occ(T_j, P)$. It

[1] In their paper, they claimed a searching time of $O(m + \frac{c^{k+d}}{(k+d)!} \log^{k+d} n \log\log n + occ)$ where c is a constant. We believe that their searching time can also be expressed in the way we specified. In addition, they claimed only the case where pattern has exactly d wildcards, and we believe the index can also support patterns with less than d wildcards.

is useful to observe the following upper bounds of γ. First, since $occ(T_j, P) \leq m$, $\gamma \leq m(\ell+1)$. Next, we give a tighter bound depending on how many repeating prefixes the text contains. For two strings X and Y, let $pre(X, Y) = 1$ if X is a prefix of Y, and 0 otherwise. We define the prefix-complexity of a text T to be $pc(T) = \max_{1\leq j\leq \ell+1} \sum_{1\leq i\leq \ell+1} pre(T_i, T_j)$. It is easy to observe that $\gamma \leq m \cdot pc(T) \leq m(\ell+1)$.

C. Wildcards in text and pattern. Given a text T with ℓ groups of wildcards. We can build an $O(n)$ space index so that, for any pattern P with h groups of wildcards, searching P takes $O(m \log n + h\beta + \gamma + occ)$ time, where $\beta = \min_{1\leq i\leq h+1} occ(P_i, T)$ and $\gamma = \sum_{j=1}^{\ell+1}(occ(T_j, P_1) + occ(T_j, P_2) + \ldots + occ(T_j, P_{h+1}))$. Note that the inequality $\gamma \leq m(\ell+1)$ still holds.

We further consider allowing a wildcard to match any single character as well as no character. We call such a wildcard an optional wildcard. Note that a group of x consecutive wildcards can match zero to x characters. An earlier work of Rahman et al. showed that an $O(n)$ space index of a text (containing no wildcard) can support checking the existence of a match for a pattern containing optional wildcards in $O(m + \alpha \log\log n)$ time where α is defined as before.

In the case of optional wildcards, it can happen that P matches both $T[s..t_1]$ and $T[s..t_2]$ where $t_1 > t_2$. Since all these matches are similar, to avoid redundency, we modify the definition of a match to include all of these matches in a single match. Precisely, a *match* is a position s in T such that P matches $T[s..t]$ for some $t \geq s$. In other words, s is the starting position of a substring of T that matches P. This definition will be used throughout the paper.

In this paper, we give space efficient indexes for optional wildcards that allows searching for all matches of P in T in three different settings namely, P contains optional wildcards, T contains optional wildcards and both P and T contains optional wildcards.

D. Optional wildcards in pattern only. Given a text T without wildcard. We can build an $O(n)$ space index so that, for any pattern P with h groups of totally g wildcards, searching for P takes $O(m + gh\beta)$ time, where $\beta = \min_{1\leq i\leq h+1} occ(P_i, T)$.

E. Optional wildcards in text only. Given a text T with ℓ groups of totally k wildcards. We can build an $O(n)$ space index so that, for any pattern P without wildcard, searching takes $O(m^2 \log n + m \log^2 n + \gamma \log n + occ)$ time, where $\gamma = \sum_{j=1}^{\ell+1} occ(T_j, P)$. We also give an $O(n \log n)$ space index so that searching takes $O(m \log^2 n + \gamma \log n + occ)$ time.

F. Optional wildcards in text and pattern. Given a text T with ℓ groups of wildcards. We can build an $O(n)$ space index so that, for any pattern P with h groups of totally $g = \sum_{i=1}^{h} g_h$ wildcards, searching P takes $O(m^2 \log n + m \log^2 n + gh\beta + \gamma \log n + occ)$ time, where $\beta = \min_{1\leq i\leq h+1} occ(P_i, T)$ and $\gamma = \sum_{j=1}^{\ell+1}(occ(T_j, P_1) + occ(T_j, P_2) + \ldots + occ(T_j, P_{h+1}))$. We also give an $O(n \log n)$ space index so that searching takes $O(m \log^2 n + gh\beta + \gamma \log n + occ)$ time.

The following table summarizes the previous and new results in different settings.

Setting	Space	Time	
Wildcards in pattern only	$O(n \log^g n)$	$O(m + 2^g \log\log n + occ)$	[3]
	$O(n)$	$O(m + \alpha)$	[6]
	$O(n)$	$O(m + h\beta)$	†
Optional wildcards in pattern only	$O(n)$	$O(m + gh\beta)$	†
Wildcards in text only	$O(n \log^k n)$	$O(m + \log^k n \log\log n + occ)$	[3]
	$O(n)$	$O(m \log n + \gamma + occ)$	†
Optional wildcards in text only	$O(n)$	$O(m^2 \log n + m \log^2 n + \gamma \log n + occ)$	†
	$O(n \log n)$	$O(m \log^2 n + \gamma \log n + occ)$	†
Wildcards in both text and pattern	$O(n \log^{g+k} n)$	$O(m + 2^g \log^k n \log\log n + occ)$	[3]
	$O(n)$	$O(m \log n + h\beta + \gamma + occ)$	†
Optional wildcards in both text and pattern	$O(n)$	$O(m^2 \log n + m \log^2 n + gh\beta + \gamma \log n + occ)$	†
	$O(n \log n)$	$O(m \log^2 n + gh\beta + \gamma \log n + occ)$	†

Notations: $\alpha = \sum_{1 \le i \le h+1} occ(P_i, T)$, $\beta = \min_{1 \le i \le h+1} occ(P_i, T)$, $\gamma = \sum_{\substack{1 \le i \le h+1 \\ 1 \le j \le \ell+1}} occ(T_j, P_i)$,

† = our result

We remark that our index can be combined with the index of Cole et al.[3] to form an index which occupies $O(n \log^d n)$ space and avoids the $\Theta(n)$ worst-case searching time if the number of wildcards in pattern does not exceed d. We will discuss the details in the last section.

2 Preliminaries

2.1 Suffix Trees and Auxiliary Data Structures

Consider a text $T[1..n]$ with some wildcards. Denote $T = T_1\phi^{k_1}T_2\phi^{k_2}\cdots\phi^{k_\ell}T_{\ell+1}$, where ϕ^{k_i} is a group of $k_i \ge 1$ wildcards, and each T_i is a *text segment* containing no wildcard. In the case where T contains no wildcard, then $T = T_1$. Similarly, we denote a pattern $P = P_1\phi^{g_1}P_2\phi^{g_2}\cdots\phi^{g_h}P_{h+1}$, where each P_i is a *pattern segment* containing no wildcard. Again, if P contains no wildcard, then $P = P_1$. As mentioned in the introduction, we assume that a wildcard can match any single character, and an optional wildcard can match any character as well as no character.

For each text segment T_i, we append a distinct character '$\$_i$' which does not appear anywhere in T. Let U be the set of all suffixes of $T_i\$_i$ for all $1 \le i \le \ell+1$. A generalized suffix tree $\mathcal{ST}$ is a compact trie comprising all strings in U [5]. Note that every edge in $\mathcal{ST}$ represents a certain substring of some T_i, and $\mathcal{ST}$ occupies $O(n)$ space. Below a *location* in $\mathcal{ST}$ refers to either a node or somewhere inside an edge, representing a prefix of a suffix in U. To ease later discussion, for each suffix of T_j, if the location in $\mathcal{ST}$ representing $T_j[i..|T_j|]$ is inside an edge, we modify $\mathcal{ST}$ to convert this location to a node (with only one child) and splitting the edge into two edges. We introduced $O(n)$ more nodes, hence $O(n)$ extra space.

Pattern matching. Given a string P without wildcard, we can use $O(|P|)$ time to determine the location in $\mathcal{ST}$ representing P in the sense that the descendant leaves of this location correspond uniquely to all strings in U that have P as a prefix. Furthermore, suppose we use $O(n)$ extra space to store on each node of $\mathcal{ST}$ the number of descendant leaves, then we have the following lemma.

Lemma 1. *Given $\mathcal{ST}$, it takes $O(|P|)$ time to compute t, the total number of matches of P in all T_j. We can also report all these matches using an addition of $O(t)$ time.*

Dictionary Matching. The suffix tree $\mathcal{ST}$ can be used to answer another query. Given a pattern $P[1..m]$ with no wildcard, we ask for the matches of each text segment T_j in P. In other words, we want to find, for all T_j, all positions i such that the substrings $P[i\,..\,i+|T_j|-1]$ equals T_j. We will add some extra information to $\mathcal{ST}$ to support this query. For each T_j, let v be the node in $\mathcal{ST}$ representing T_j. We *mark* at v with a label T_j. For every node in $\mathcal{ST}$, we store a pointer *up* to the nearest marked ancestor. A location in $\mathcal{ST}$ maybe marked more than once; in this case we store in the location a list of all the labels. All the node splitting and marking information take $O(n)$ space. To perform a dictionary matching for a pattern $P[1..m]$ with no wildcard, we first locate, for each $1 \leq i \leq m$, the location v_i in $\mathcal{ST}$ representing the longest prefix of $P[i..m]$, then traverse repeatedly the *up* pointers from v_i until all marked ancestors of v_i are visited. Each marked ancestor T_j represents a match of T_j with $P[i\,..\,i+|T_j|-1]$. By using suffix links [3], it takes only $O(m)$ time to locate all v_i.

Lemma 2. *Given $\mathcal{ST}$, for any pattern P with no wildcard, it takes $O(m+t)$ time to locate all matches of T_j in P for all $1 \leq j \leq \ell+1$, where t is the number of matches reported.*

2.2 Auxiliary Data-Structure for Fast String Comparison

We give a numbering scheme to assign a subinterval of $[1, 16n]$ to each node in $\mathcal{ST}$ so as to facilitate a certain kind of string searching. The subinterval is called the *span* of the node. Below we show an algorithm to assign one or more integers to each node and edge in $\mathcal{ST}$. We begin with a counter c that is initially zero. We perform a depth first search on $\mathcal{ST}$ starting from the root, visiting the leaves of $\mathcal{ST}$ in its lexicographical order. Every time when we visit an edge or a node, we first increment c and then assign c to the edge or the node. Each internal node v is assigned at least 2 numbers. If a and b are the smallest and largest numbers assigned, we call the range $[a, b]$ the span of v. Each edge e is assigned exactly 2 numbers, say, p and q where $p < q$. We call the range $[p, q]$ the span of e. Each leaf u is assigned exactly one number. We call it the *leaf number* of u. The span information can all be stored using $O(n)$ extra space.

We now define the span for any string X as follows. Let v be the location in $\mathcal{ST}$ representing the longest prefix of X, and let $[p, q]$ be the span of the node or edge where v resides. If v represents the whole string X (i.e., X is a substring of some T_i), we define the span of X to be $[p, q]$, and $[p, p]$ otherwise.

Given a text $T[1..n]$, if we have built $\mathcal{ST}$ with the span information, then we can perform the following string comparison: Given the span $[p, q]$ of a string X and the span $[d, e]$ of any suffix Y of some T_j, determine whether X is a prefix of Y and whether Y is a prefix of X. The lemma below states that this comparison can be done in $O(1)$ time.

Lemma 3. *Given the span $[p, q]$ of a string X and the span $[d, e]$ of a suffix Y of some T_j, we have:* (1) *X is a prefix of Y if and only if $p \leq d \leq e \leq q$, and* (2) *Y is a prefix of X if and only if $d \leq p \leq q \leq e$.*

To exploit Lemma 3 in our solution for wildcard matching, we will precompute the span of every suffix of T_j and uses $O(n)$ extra space to store these span information. Then, given a pattern P, it takes $O(m)$ time to compute the span of every suffix of P. Afterwards we can compare any suffix of P and any suffix of T_j in $O(1)$ time.

We can extend the above lemma to support comparing any prefix of P with any prefix of some T_j in $O(1)$ time. We can achieve this by storing a generalized suffix tree $\mathcal{ST}'$ comprising the suffixes of the reverse of every $\$_i T_i$, and the span information on $\mathcal{ST}'$. The spans of every suffix of the reverse of T_j and every suffix of the reverse of P are computed similarly.

2.3 Orthogonal Segment Intersection

Given a set H of n horizontal line segments in the Cartesian plane. We want to find efficiently all segments in H that intersect with a given vertical line segment.

Lemma 4. [1] *We can build an $O(n)$ space data structure for H such that when a vertical line segment is given, we can compute t, the number of intersecting horizontal segments in H, in $O(\log n)$ time, and report all these horizontal segments in $O(t)$ time.*

3 Wildcards in Pattern Only

As a warm-up, this section considers the simplest setting where the text T contains no wildcards, and wildcards can only appear in a pattern. In this case, the $O(n)$-space data structures described in Section 2 (i.e., the suffix tree $\mathcal{ST}$ and the auxiliary information about the number of descendant leaves and span) are already sufficient to support efficient pattern matching.

Below we will first consider wildcards that each must match a character, then we study the optional wildcards. Suppose that we are given a pattern P of length m, containing $h \geq 0$ groups of consecutive wildcards, i.e., $P = P_1\phi^{g_1}P_2\phi^{g_2}\cdots\phi^{g_h}P_{h+1}$. First of all, using $O(m)$ time, we find the spans of $P_1, P_2, \ldots, P_{h+1}$. Then we make use of these spans to find all matches of P in T (i.e. all positions i such that $T[i\,..\,i+|P|-1] = P$) in $O(h\beta)$ time. Details of the matching algorithm are as follows. It first finds a set of candidate positions i in T which is a superset of all matches of P in T, then it verifies whether each of them is really a match.

Lemma 5. *We can verify in $O(h)$ time if a candidate position i gives a match of P in T.*

Proof. Let $s_1 = i$, $s_2 = s_1 + |P_1| + g_1, \ldots,$ and $s_{h+1} = s_h + |P_h| + g_h$. Note that i is a match of P in T if $T[s_1 .. s_1 + |P_1| - 1] = P_1$, $T[s_2 .. s_2 + |P_2| - 1] = P_2$, ..., and $T[s_{h+1} .. s_2 + |P_{h+1}| - 1] = P_{h+1}$. Recall that the span of each P_i is known. By Lemma 3, the comparison of each P_i takes $O(1)$ time. Thus, it takes $O(h)$ time to verify a position i. □

Next, we show how to generate candidate positions that cover all matches of P in T. For any substring of T that matches P, it must contain all $P_1, P_2, \ldots, P_{h+1}$ as substrings. Thus the matches of any particular P_i in T are sufficient to be a set of candidates. The key question is which P_i to use. By Lemma 1, we can count, for all P_i, the number of matches of P_i in T (i.e., $occ(P_i, T)$) using $O(|P_1| + |P_2| + \ldots + |P_{h+1}|) = O(m)$ time. We pick i^* such that P_{i^*} generates the least number of matches. Then, by Lemma 1 again, we can locate all the matches for P_{i^*} in $occ(P_{i^*}, T)$ time, and, by Lemma 5, we verify each match in $O(h)$ time. The total time complexity is stated below.

Theorem 1. *We can build an $O(n)$-space index for a text T without wildcard so that searching all the matches of any pattern P with h groups of wildcards takes $O(m + h\beta)$ time, where $\beta = \min_{1 \le i \le h+1} occ(P_i, T)$.*

Optional Wildcards. The index described above can also deal with optional wildcards. We only need to modify the matching algorithm and, in particular, the verification of a candidate. The time required for the latter increases to $O(gh)$, where $g = \sum_{i=1}^{h} g_i$ is the total number of wildcards in P. Details are as follows. Again, we first choose a pattern segment P_i that minimizes $occ(P_i, T)$ among all pattern segments, and locate all matches of P_i in T. These matches form a set of candidates for further verification.

For each match of P_i in T, we further check whether there is also a match with $\phi^{g_i} P_{i+1} \phi^{g_{i+1}} \ldots \phi^{g_h} P_{h+1}$, as well as a match with $P_1 \phi^{g_1} \ldots \phi^{g_{i-2}} P_{i-1} \phi^{g_{i-1}}$. Both checking can be done in a simple iterative manner. Let us consider the tail first. Suppose there is a match of P_i at position s of T, i.e., $P_i = T[s .. s + |P_i| - 1]$. Intuitively, we want to skip some $c_i \in [0, g_i]$ characters starting from $s + |P_i|$, and check whether P_{i+1} can match at position $s_{i+1} = s .. s + |P_i| + c_i$. Such checking has to be done for every $c_i \in [0, g_i]$ and takes $O(g_i)$ time. Let W_i be the set of all c_i's that pass this checking. If W_i is non-empty, we proceed to verify P_{i+2}. This time we want to check weather there is a match for P_{i+2} at position $s_{i+2} = s + |P_i| + |P_{i+1}| + c_{i+1}$, where $c_{i+1} = x + y$ for some $x \in W_i$ and $y \in [0, g_{i+1}]$. Note that $0 \le c_{i+1} \le g_i + g_{i+1}$, and the checking due to P_{i+2} takes $O(g_i + g_{i+1})$ time. We repeat this procedure until there is no match for some P_j where $j > i$ or P_{h+1} has been verified. The time required is $O(g_i + (g_i + g_{i+1}) + (g_i + g_{i+1} + g_{i+2}) + .. + \sum_{j=i}^{h} g_j) = O(gh)$.

The checking for $P_1 \phi^{g_1} \ldots \phi^{g_{i-2}} P_{i-1} \phi^{g_{i-1}}$ can be done in a similar way. We start with P_{i-1} and work backward to P_1. The time required is also $O(gh)$. After we have matched P_1, we record the position in T that matches P_1. Note that there are $\beta = \min_{1 \le i \le h+1} occ(P_i, T)$ candidates to verify and the time required is $O(gh\beta)$. We note that the same position can be recorded multiple times, and

we need to remove the duplicates. Using constant time initialization technique[2] on an array of size n, duplicates are removed in $O(gh\beta)$ time.

Theorem 2. *We can build an $O(n)$-space index for a text T without wildcard so that searching all the matches for any pattern P with h groups of totally g optional wildcards, takes $O(m + gh\beta)$ time.*

4 Wildcards in Text Only

When the text is allowed to contain wildcards, building a space-efficient index to support efficient pattern matching is no longer trivial even if patterns have no wildcards. This section and the next section consider this setting for (ordinary) wildcards and optional wildcards, respectively.

Suppose T is a text with ℓ groups of wildcards, i.e. $T = T_1\phi^{k_1}T_2\phi^{k_2}\cdots\phi^{k_\ell}T_{\ell+1}$. Given a pattern P without wildcard, we classify each substring of T that matches P into one of the followings.

Type 1. It is a substring of some T_j, $1 \le j \le \ell+1$.
Type 2. It matches $T[s..t]$ which contains exactly one wildcard group ϕ^{k_j} where $1 \le j \le \ell$.
Type 3. It matches $T[s..t]$ which contains more than one wildcard groups.

Let occ_1, occ_2, occ_3 be the number of Type 1, Type 2, and Type 3 matches, respectively. Note that $occ = occ_1 + occ_2 + occ_3$. Given $\mathcal{ST}$, Type 1 matches can be found in $O(m + occ_1)$ time by Lemma 1. In the next two subsections, we will show how to build extra data structures to support finding Type 2 and Type 3 matches efficiently.

4.1 Finding Type 2 Matches

To find Type 2 matches efficiently, we define the notion of a *split* of T. We will show that there are at most $O(n)$ splits. More interestingly, we can preprocess these splits using the suffix tree $\mathcal{ST}$ and represent them as horizontal line segments. We further build an index for orthogonal segment intersection for these splits. Then finding Type 2 matches can be reduced to a query in the form of a vertical segment to the latter index.

If P matches a substring of T containing exactly one wildcard group, then P is a substring of $T_j\phi^{k_j}T_{j+1}$ for some $1 \le j \le \ell$.

Definition 1. *The d-split of $T_j\phi^{k_j}T_{j+1}$ is a pair of strings (X, Y) such that X is a suffix of T_j, $Y = T_{j+1}$, and $|X| + k_j = d$. The set of d-splits of T is the set of d-split of $T_j\phi^{k_j}T_{j+1}$ for all $j = 1, 2, \ldots, \ell$.*

Suppose that there is a Type 2 match of P inside $T_j\phi^{k_j}T_{j+1}$. Then for some $d \le m$, the d-split (X, Y) of $T_j\phi^{k_j}T_{j+1}$ satisfies the relation that X is a prefix of $P[1\,..\,d]$, and $P[d+1\,..\,m]$ is a prefix of Y. The following lemma is the key to finding Type 2 matches.

Lemma 6. *Given a text $T[1..n]$, we can build an $O(n)$-space data structure to store all splits of T. Then, given an integer d and a string $Q[1..q]$, suppose we know the span of $Q[1..d]$ and $Q[d+1..q]$ in $\mathcal{ST}$, we can find in $O(\log n)$ time the number of d-splits (X, Y) such that X is a prefix of $Q[1..d]$ and $Q[d+1..q]$ is a prefix of Y. Furthermore, if there are t such d-splits, we can report in $O(t)$ time the matches of Q corresponding to these d-splits.*

Proof. We reduce the problem to orthogonal segment intersection. For every $d \in [1, n]$, we build an orthogonal segment intersection data structure $\mathcal{OSI}_d$ which contains one horizontal line for each d-split (X, Y) of $T_j\phi^{k_j}T_{j+1}$. For a d-split (X, Y), let y be the leaf number of the leaf in $\mathcal{ST}$ representing $Y\$$, and let $[x_1, x_2]$ be the span of the string X in $\mathcal{ST}$; we put a horizontal segment (x_1, y)—(x_2, y) in $\mathcal{OSI}_d$. Note that any string having X as a prefix has a span that is enclosed by $[x_1, x_2]$ and any string that is a prefix of Y has a span that encloses y.

For any given string Q, let $[x'_1, x'_2]$ be the span of $Q[1..d]$ and $[y'_1, y'_2]$ be the span of $Q[d+1..q]$. If $Q[1..d]$ is a prefix of X, then $[x'_1, x'_2]$ is enclosed by $[x_1, x_2]$. If $Q[d+1..q]$ is a prefix of Y, $[y'_1, y'_2]$ encloses y. By querying for horizontal segments in $\mathcal{OSI}_d$ that intersect the vertical segment (x'_1, y'_1)—(x'_1, y'_2), we can find all d-split (X, Y) such that X is a prefix of $Q[1..d]$ and $Q[d+1..q]$ is a prefix of Y. □

We are ready to show how to find all Type 2 matches of a pattern $P[1..m]$. For every $2 \le d \le m$, we use Lemma 6 to find all d-splits matching $P[1..m]$. Each of them is reported as a Type 2 match. Now we analyze the space and time complexity of the above data structure and searching algorithm. By definition, each $T_j\phi^{k_j}T_{j+1}$ contributes exactly one (k_j+x)-split for all $x \in [0, |T_j|]$. Since every $T_j\phi^{k_j}T_{j+1}$ contributes a total of $|T_j|+1$ splits, there are totally $O(n)$ splits for text $T[1..n]$, and the data structure requires $O(n)$ space. By Lemma 4, each orthogonal segment intersection query takes $O(\log n+t)$ time to report t matches. The search makes m queries for $P[1..d]$ and $P[d+1..m]$ for every $1 \le d \le m$, where each Type 2 match is reported exactly once.

Lemma 7. *We can locate all Type 2 matches in $O(m \log n + occ_2)$ time.*

4.2 Finding Type 3 Matches

Now we consider the remaining case where P matches a substring of T containing more than one wildcard group. It is easy to see that if P matches at least two wildcard groups, then P has some T_j as a substring. We will use all these T_j which are matches in P as a basis to find Type 3 matches. Using Lemma 2, we can find all matches of T_j in P. Now we need to check whether we can combine these matches to form Type 3 matches.

Suppose we know that $T_j = T[s..t]$ is a prefix of $P[i..m]$. If T_j forms part of a Type 3 match, then this match starts at position $s-i+1$ in T. We first determine these candidate positions from all matching T_j. Then we can verify each candidate position whether it gives a Type 3 match. We declare an array $A[1..n]$, where $A[r] > 0$ indicates r is candidate position. First, we initialize all

elements as zero. For each $T_j = T[s..t]$ being a prefix of $P[i..m]$, we increment $A[s-i+1]$ by 1. Now $A[r]$ stores the number of times it is marked as a candidate position. We remain to verify these candidate positions, i.e. all r where $A[r] > 0$.

Let $a_j = 1 + \sum_{i=1}^{j-1}(|T_i| + k_i)$ denote the starting position of T_j in T for all $1 \leq j \leq \ell+1$. We perform the following steps for each r where $A[r] > 0$. Pick the smallest j_1 such that $a_{j_1} \geq r$ and the largest j_2 such that $a_{j_2} + |T_{j_2}| \leq r+m$. We have chosen j_1 and j_2 such that $T_{j_1}, T_{j_1+1}, \ldots, T_{j_2}$ are the only text segments that are fully contained in $T[r\,..\,r+m-1]$, i.e. text segments that should be a substring of P. By comparing $A[r]$ with $j_2 - j_1 + 1$, we can check whether every of $T_{j_1}, T_{j_1+1}, \ldots, T_{j_2}$ has been found as a substring of P at the correct position. What remains is to verify whether the two ends of P matches the incomplete parts of T_{j_1-1} and T_{j_2+1} that fall into $T[r\,..\,r+m-1]$, i.e., whether they match the corresponding prefix and suffix of P. Precisely, a position r in T is verified to be a match of P if (1) $A[r] = j_2 - j_1 + 1$, (2) $T_{j_2+1}[1\,..\,r+m-a_{j_2+1}]$ is a suffix of P when $a_{j_2} + k_x \leq r + m - 1$, and (3) $T_{j_1-1}[a_{j_1-1} - r\,..\,|T_{j_1-1}|]$ is a prefix of P when $a_{j_1} - k_{j_1-1} \geq r + 1$.

Lemma 8. *Type 3 matches can be located in $O(m+\gamma)$ time, $\gamma = \sum_{j=1}^{\ell+1} occ(T_j, P)$.*

Proof. Locating all matches of T_j in P takes $O(m+\gamma)$ time. We use the constant time initialization technique again[2] so that the array is initialized on demand in amortized $O(1)$ time per cell. When we want to loop over all $A[r] > 0$, we can simply scan each of the γ matches. For each r, we verify the above conditions by Lemma 3. We mark $A[r] = 0$ after we are done so as to avoid r being processed again. □

Combining Lemmas 7 and 8, we have the following theorem.

Theorem 3. *We can build an index for a text T containing ℓ groups of wildcards in $O(n)$ space so that for any pattern P without wildcard, searching takes $O(m \log n + \gamma + occ)$ time where $\gamma = \sum_{1 \leq i \leq \ell+1} occ(T_i, P)$ and occ is the number of matches of P in T.*

When the pattern P contains no wildcard and the text T contains optional wildcards. The problem can be solved similarly by classifying the matches into three types. The detail will be described in the full version. We have the following theorem.

Theorem 4. *We can build an index for a text T containing ℓ groups of wildcards in $O(n)$ space so that for any pattern P without wildcard, searching takes $O(m^2 \log n + m \log^2 n + \gamma \log n + occ)$ time where $\gamma = \sum_{1 \leq i \leq \ell+1} occ(T_i, P)$ and occ is the number of matches of P in T. If $O(n \log n)$ space is given, searching time is reduced to $O(m \log^2 n + \gamma \log n + occ)$.*

5 Wildcards in Both Pattern and Text

Now we consider the string matching problem where both the text and the pattern contains wildcards, for both types of wildcards. We basically combine

the results in the previous sections. Similar to Section 2, we evaluate all pattern segments and picks the one which generates the fewest candidates in T, where a candidate is a match of the pattern segment in T. Then we retrieve all the candidates and verify each of them against other pattern segments.

Let us first consider (ordinary) wildcards. We can count the number of candidates for each pattern segment P_i as follows. Lemmas 1 and 7 allow us to count the number of Type 1 and Type 2 matches using $O(|P_i|)$ and $O(|P_i| \log n)$ time respectively. We can also retrieve all Type 3 matches of P_i using $O(|P_i| + \sum_{j=1}^{\ell+1} occ(T_j, P_i))$ time. Now we pick P_{i^*} and retrieving all its candidates similar to Section 3. Lemma 9 allows us verify every other pattern segment in $O(1)$ time. and hence we can verify each candidate in $O(h)$ time. Thus, we have Theorem 5.

Lemma 9. *Suppose we have already computed Type 3 matches of every P_i. Given a pattern segment P_i, we can determine in $O(1)$ time whether it matches $T[s..t]$.*

Theorem 5. *We can build an index for a text T with ℓ groups of wildcards in $O(n)$ space so that for any pattern P with h groups of wildcards, searching takes $O(m \log n + \gamma + h\beta)$ time where $\beta = \min_{1 \le \imath \le h+1} occ(P_i, T)$ and $\gamma = \sum_{j=1}^{\ell+1}(occ(T_j, P_1) + occ(T_j, P_2) + \cdots + occ(T_j, P_{h+1}))$.*

For optional wildcards, as shown in the full version, we have the following theorem.

Theorem 6. *We can build an index for a text T with ℓ groups of optional wildcards in $O(n)$ space so that for any pattern P with h groups of totally g optional wildcards, searching takes $O(m^2 \log n + m \log^2 n + \gamma \log n + gh\beta + occ)$ time where $\beta = \min_{1 \le \imath \le h+1} occ(P_i, T)$ and $\gamma = \sum_{j=1}^{\ell+1}(occ(T_j, P_1) + \cdots + occ(T_j, P_{h+1}))$. If $O(n \log n)$ space is given, searching time is reduced to $O(m \log^2 n + \gamma \log n + gh\beta + occ)$.*

6 More Space for Pattern with Wildcards

For the cases where the pattern P contains (ordinary) wildcards, we note that the time complexity includes a term containing β. Since β is the minimum of $occ(P_i, T)$ over all P_i, it can be very large when compared with the other terms. We give a trade-off solution which allows using more space to decrease the dependency on β while slightly increasing other terms of the searching time.

Recall that our algorithm in Section 3 first locates all "candidates" of some P_{i^*} by the suffix tree, and then verify each candidate with all other pattern segments. We can do better if we have an index more powerful than the suffix tree. Consider a substring $X\phi Y$ of P where ϕ is a wildcard. We note that $occ(X\phi Y, T)$ can be much smaller than both $occ(X, T)$ and $occ(Y, T)$. The index by Cole et al.[3] as mentioned in the introduction can report all matches of $X\phi Y$ efficiently. Using this index to generate candidates, we only need to verify $occ(X\phi Y, T)$ candidates.

Given an integer d, let Ψ_d be the set of all maximal substrings of P such that (1) it contains at most d wildcards, and (2) neither of its first and last characters are wildcard. We can build the index of Cole et al. for T using $O(n \log^d n)$ space, such that we can pick $Q \in \Psi_d$ that gives the smallest $occ(Q, P)$ using $O(m + |\Psi_d| 2^{\min\{g,d\}} \log\log n)$ time, and locate its matches in $O(2^{\min\{g,d\}} \log\log n + occ)$ time. As a result, candidate generation takes $O(|\Psi_d| 2^{\min\{g,d\}} \log\log n)$ time and verification takes $O(|\Psi_d|\beta_d)$ time. That is, we can build an index using $O(n \log^d n)$ space so that searching takes $O(m + |\Psi_d| 2^{\min\{g,d\}} \log\log n + |\Psi_d|\beta_d)$ time, where $\beta_d = \min_{Q \in \Psi_d} occ(Q, T)$.

Unlike Cole's index which only works when $g \leq d$, our index works for patterns with any number of wildcards. However, when $g \leq d$, since $|\Psi_g| = 1$ and $\beta_d = occ$, searching in our index takes $O(m + 2^g \log\log n + occ)$ time, matching both the space and time complexity of Cole's index. If $g > d$, we will have more candidates than occ. Let $\beta_d = \min_{Q \in \Psi_d} occ(Q, T)$. We observe that $\beta_g \leq \beta_{g-1} \leq \ldots \leq \beta_1 \leq \beta$, since every string X in Ψ_d has a longer counterpart in Ψ_{d+1}. In other words, X generates at least the number of candidates Y generates. This means a larger d will result in fewer candidates, but we need more space for indexing and more time to search for candidates.

References

1. Boroujerdi, A.R., Moret, B.M.E.: Persistence in computational geometry. In: Proceedings of 7th Canadian Conference on Computational Geometry, pp. 241–246 (1995)
2. Briggs, P., Torczon, L.: An Efficient Representation for Sparse Sets. ACM Letters on Programming Languages and Systems 2, 59–69 (1993)
3. Cole, R., Gottlieb, L.A., Lewenstein, M.: Dictionary matching and indexing with errors and don't cares. In: Proceedings of Symposium on Theory of Computing, pp. 91–100 (2004)
4. Fischer, M.J., Paterson, M.S.: String matching and other products. In: Complexity of computation, vol. 7, pp. 113–125. Massachusetts Institute of Technology, Cambridge, MA (1974)
5. Gusfield, D.: Algorithms on strings, trees, and sequences: computer science and computational biology. Cambridge University Press, New York (1997)
6. Iliopoulos, C.S., Rahman, M.S.: Pattern Matching Algorithms with Don't Cares. In: Proceedings of Student Research Forum of SOFSEM, pp. 116–126 (2007)
7. McCreight, E.M.: A Space-economical Suffix Tree Construction Algorithm. Journal of the ACM 23(2), 262–272 (1976)
8. Rahman, M.S., Iliopoulos, C.S., Lee, I., Mohamed, M., Smyth, W.F.: Finding Patterns with Variable Length Gaps or Dont Cares. In: Proceedings of Annual International Computing and Combinatorics Conference, pp. 146–155 (2006)
9. Weiner, P.: Linear Pattern Matching Algorithms. In: Proceedings of Symposium on Switching and Automata Theory, pp. 1–11 (1973)

2-Stage Fault Tolerant Interval Group Testing

Ferdinando Cicalese[1] and José Augusto Amgarten Quitzau[1,2]

[1] AG Genominformatik, Technical Faculty, Bielefeld University, Germany
[2] International Graduate School in Bioinformatics and Genome Research, Center for Biotechnology, Bielefeld University, Germany

Abstract. We study the following fault tolerant variant of the interval group testing model: Given four positive integers n, p, s, e, determine the minimum number of questions needed to identify a (possibly empty) set $P \subseteq \{1, 2, \ldots, n\}$ ($|P| \leq p$), under the following constraints. Questions have the form "Is $I \cap P \neq \emptyset$?", where I can be any interval in $\{1, 2 \ldots, n\}$. Questions are to be organized in s batches of non-adaptive questions (stages), i.e, questions in a given batch can be formulated relying only on the information gathered with the answers to the questions in the previous batches. Up to e of the answers can be erroneous or lies.

The study of interval group testing is motivated by several applications. remarkably, to the problem of identifying splice sites in a genome. In particular, such application motivates to focus algorithms that are fault tolerant to some degree and organize the questions in few stages, i.e., on the cases when s is small, typically not larger than 2. To the best of our knowledge, we are the first to consider fault tolerant strategies for interval group testing.

We completely characterize the fully non-adaptive situation and provide tight bounds for the case of two batch strategies. Our bounds only differ by a factor of $\sqrt{11/10}$ for the case $p = 1$ and at most 2 in the general case.

1 Introduction

Problem Statement. In this paper we consider fault tolerant algorithms for interval group testing. An instance of the problem is given by four non-negative integers n, p, s, e and a subset $P \subseteq O = \{1, 2, \ldots, n\}$, such that $|P| \leq p$. The set O is the search space and P is the set of *positive* objects that have to be identified. Queries are binary test asking "Is $P \cap \{i, i+1, \ldots, j\} \neq \emptyset$?", for some $1 \leq i \leq j \leq n$. The target is to identify P by using the minimum possible number of queries. We assume that tests are arranged in *stages*: in each stage a certain number of tests is performed non-adaptively, while tests of a given stage can be determined depending on the outcomes of the tests in all previous stages. Finally, we assume that up to a finite number e of the answers might be erroneous or lies.

For each value of the parameters n, p, s, e we want to determine $\mathcal{N}(n, p, s, e)$, the worst-case number of tests that are necessary (and sufficient) to successfully identify all positives in a search space of cardinality n, under the hypothesis that

T. Tokuyama (Ed.): ISAAC 2007, LNCS 4835, pp. 858–868, 2007.

the number of positives is at most p, s-stage algorithms are used and up to e answers are lies.

Motivations and Related Research. Group testing is a basic paradigm in the theory of combinatorial search and is efficiently used in very diverse areas such as quality control, multiple access communication, computational molecular biology, data compression, and data streams algorithms among the others (see [5,6,9,14,17,3]). Group testing with interval tests also arises in variety of domains, e.g., detecting holes in a gas pipe [5,4], finding faulty links in an electrical or communication network, data gathering in sensor networks [10,11,12], just to mention a few.

Our main motivation for the study of interval group testing comes from its application to the problem of determining exon-intron boundaries within a gene [15,18]. In a very simplified model, a gene is a collection of disjoint substrings within a long string representing the DNA molecule. These substrings, called *exons*, are separated by substrings called *introns*. The boundary point between an exon and an intron is called a *splice site*, because introns are spliced out between transcription and translation. Determining the splice sites is an important task, e.g., when searching for mutations associated with a gene responsible for a disease.

In [18], a new experimental protocol is proposed that searches for the exons boundaries using group testing. This consists of selecting two positions in the cDNA, a copy of the original genomic DNA from which introns have been spliced out, and determine whether they are at the same distance as they were in the original genomic DNA string. If these distances do not coincide then at least one intron (and hence a splice site) must be present in the genomic DNA between the two selected positions. The formulation of splice sites identification as a group testing problem with interval queries is explicitly stated in [13,15,18]. The advantages of splice site detection by distance measurements over sequence-based methods using, e.g., Hidden Markov Models are that this method works without expensive sequencing of genomic DNA and it gives the results directly from experiments, without relying on inference rules. The work [18] and the book [15] report about the experimental evaluation, on real data, of the algorithm ExonPCR, that finds exon-intron boundaries within a gene. The authors of [18] give also a simple asymptotic analysis of their $\Theta(\log n)$-stage algorithm. The question was whether there exist less obvious but more efficient query strategies for Interval Group Testing, and more importantly, algorithms able to cope with the technical limitation of the experiments, and particularly with errors. We remark that non-adaptive strategies are desirable in this context, in order to avoid long waiting periods necessary to prepare each experiment. However a totally non-adaptive algorithm (with $s = 1$) needs unreasonably many queries. Thus, the necessity arises to trade more stages for fewer queries, but without exceeding with stages. In [1] the first rigorous algorithmic study of the problem was presented, and for the case $s \leq 2$ a precise evaluation of $\mathcal{N}(n, p, s, 0)$ was given. In [2] a sharper asymptotic estimation of $N(n, p, s, 0)$ was given that is optimal up to the constant of the main term in the case of large s.

The necessity of dealing with errors in the tests had been already stated in the seminal papers [15,18] and reaffirmed in the subsequent ones. However, to the best of our knowledge, ours are the first non trivial results on this interesting variant of the problem.

Our Results. We focus on strategies that use adaptiveness at most once, i.e., strategies with questions organized in one or two batches of non adaptive queries ($s \in \{1, 2\}$). In fact, according to [7] *... the technicians who implement the pooling strategies generally dislike even the* 3*-stage strategies that are often used [...]. The pools are either tested all at once or in a small number of stages (usually at most 2).* We exactly determine $N(n, p, 1, e)$ and provide very tight bounds for the $N(n, p, 2, e)$ that in the case $p = 1$ at most differ by a factor of $\sqrt{11/10}$, and at most by a factor 2 in all the other cases. We remark that these are the first non trivial results on fault tolerant interval group testing procedures and we stress the necessity to drive attention onto the fault tolerant variant of interval group testing.

2 Definitions and Notation

In this section we fix the notation used in the text. The set of objects where we try to find the positives is the set of the first n non-negative integers $[n] = \{1, 2, \ldots, n\}$. By abuse of notation we shall use square brackets to denote intervals of integers in $[n]$. Then, for each $1 \leq i \leq j \leq n$, we shall use $[i, j]$ to denote the set $\{i, i + 1, \ldots, j\}$. Given an interval $\pi = [i, j]$, we shall denote its size by $|\pi|$, i.e., $|\pi| = j - i + 1$. By definition each query asks about the intersection of a given interval with the set of positive elements. Therefore, we shall identify a query with the interval it specifies. We say that a query $Q \equiv [i, j]$ covers an element $k \in [n]$ if $k \in [i, j]$.

A query $Q \equiv [i, j]$ has two boundaries: the left, $(i-1, i)$, and the right, $(j, j+1)$. For the sake of definiteness, we assume that, for any a, a the query $[1, a]$ has left boundary $(0, 1)$, and the query $[a, n]$ has right boundary $(n, n+1)$. A multiset of queries $\mathcal{Q}$ defines a set of boundaries $\mathcal{B}(\mathcal{Q}) = \{(i_1, i_1 + 1), (i_2, i_2 + 1), \ldots\}$, where $i_k < i_{k+1}$. Every interval $[i_k + 1, i_{k+1}]$ is called a *piece*. Because every query has two distinct boundaries, but two queries may share some boundaries, we have $|\mathcal{B}(\mathcal{Q})| \leq 2\,|\mathcal{Q}|$. A boundary B of a piece P is said to be *turned to* the piece if there is a query Q such that $P \subset Q$ and B is also a boundary of Q. A piece is called a *2-piece* if both its boundaries are turned to it. A piece that has only one of it boundaries turned to it is called a *1-piece*. If none of the boundaries of a piece are turned to it, the piece is called a *0-piece*. Figure 1 illustrate the definitions given so far.

We shall also use the definition of a *YES set*. Given a multiset of queries $\mathcal{Q}$, a *YES set* (for $\mathcal{Q}$) is a subset of $\mathcal{Q}$ such that there exists a set of positives P ($|P| \leq p$) such that answering YES to queries in the *YES set* and NO to the other queries, the answers are consistent with P, but for at most e lies. A *YES set* is basically a possible (legal) strategy for the adversary, given the set of questions $\mathcal{Q}$. A YES set is called *specific* if the intersection of all its queries

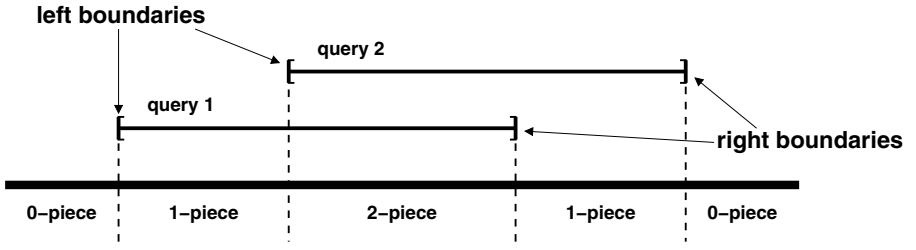


Fig. 1. A set of two interval queries, which partition the set of objects into 5 pieces. The thicker line represents the set of objects.

corresponds to a single piece, and the piece has at most one positive, otherwise it is called *unspecific*. More formally, a YES set $\mathcal{Y} \subset \mathcal{Q}$ is specific if and only if there is a piece π of $\mathcal{Q}$, with $|\pi \cap P| \leq 1$, such that $\bigcap_{Q \in \mathcal{Y}} Q = \pi$.

3 Non-adaptative Interval Group Testing with One Error

We start our analysis with the case of 1 stage strategies. In fact, the results in this section will be the basis for the analysis of the more practical two batch case. The following two theorems completely characterize 1-stage e-fault tolerant interval group testing.

Theorem 1. *For all $n \geq 1$ and $e \geq 0$, it holds that* $\mathcal{N}(n,1,1,e) = \left\lceil \frac{(2e+1)(n+1)}{2} \right\rceil$.

Proof. The lower bound directly follows from the following claim.
Claim. Every strategy that correctly identifies the (only) positive or reports $P = \emptyset$, uses a set of questions such that there are at least $2e+1$ questions' boundaries $(i, i+1)$ for each $i = 0, 1, \ldots, n$.

By contradiction, let us consider a strategy such that for some $i \in [n]$ there are $b \leq 2e$ questions with a boundary $(i, i+1)$. Let $\mathcal{Q}$ be the set of such questions and $\mathcal{Q}_1$ the set of all questions in $\mathcal{Q}$ which contain i. Assume, without loss of generality, that $|\mathcal{Q}_1| \geq |\mathcal{Q} \setminus \mathcal{Q}_1|$.

Let the adversary answer i) NO to all the questions having empty intersection with $\{i, i+1\}$, ii) YES to all questions including both i and $i+1$, iii) YES to exactly $\lceil |\frac{\mathcal{Q}}{2}| \rceil$ questions in $\mathcal{Q}_1$ and NO to the remaining ones in $\mathcal{Q}_1$, iv) answers YES to all the questions in $\mathcal{Q} \setminus \mathcal{Q}_1$.

A moment reflection shows that, due to the possibility of having up to e erroneous answers, the above set of answers is consistent with the both cases when $P = \{i\}$ and $P = \{i+1\}$[1].Hence, the given strategy cannot correctly discriminate among the above possibilities. The claim is proved.

Therefore, any strategy that is able to correctly identify P must use in total at least $(2e+1)(n+1)$ boundaries. Then, the desired results follows by observing that each question can cover at most 2 boundaries.

[1] In particular, for the cases, $i = 0$ (respectively $i = n$) the ambiguity is whether P contains no elements or the element is 1 (resp. n).

We now turn to the upper bound. Direct inspection shows that for $n \leq 3$ there exists an easy strategy with the desired number of questions.

For each $k \geq 2$, let $\mathcal{A}_{2k+1} = \{[1,2],[2,4],[4,6],\ldots,[2k-2,2k],[2k,2k+1]\}$ and $\mathcal{A}^1_{2k} = \{[2,2k-1],[3,2k-2],\ldots,[k,k+1]\}$, $\mathcal{A}^2_{2k} = \{[1,k],[k+1,2k]\}$, and $\mathcal{A}^3_{2k} = \{[1,k]\}$.

Then, for $n \geq 4$, the following strategy attains the desired bound.

If n is odd, the strategy consists of asking $2e+1$ times the questions in $\mathcal{A}_n$. These amount to $(2e+1)\lceil (n+1)/2 \rceil = \lceil (2e+1)(n+1)/2 \rceil$ questions which clearly cover $2e+1$ times each boundary $(i, i+1)$, for each $i = 0, 1, \ldots, n$.

If n is even, let $k = n/2$. Now, the strategy consists of asking $2e+1$ times the questions in $\mathcal{A}^1_n$, plus $e+1$ times the questions in $\mathcal{A}^2_n$, plus e times the questions in $\mathcal{A}^3_n$. In total, in this case, the strategy uses $(2e+1)(k-1) + 2(e+1) + e = (2e+1)k + e + 1 = \lceil (2e+1)(2k+1)/2 \rceil = \lceil (2e+1)(n+1)/2 \rceil$, as desired.

For the case of more positives we have the following generalization.

Theorem 2. *For all integers $n \geq 1, p \geq 2, e \geq 0$, it holds that $\mathcal{N}(n,p,1,e) = (2e+1)n$*

Proof. The upper bound is trivially obtained by a strategy made of $(2e+1)$ copies of the singleton questions $\{1\},\{2\},\ldots,\{n\}$.

The lower bound is obtained proceeding in a way analogous to the argument used in the previous theorem. Here, we argue that every strategy that correctly identifies P must ask, for each $i = 1, 2, \ldots, n-1$, at least $2e+1$ questions with boundary $(i, i+1)$ and including i, and at least $2e+1$ questions with boundary $(i, i+1)$ and including $i+1$. Moreover, it must ask at least $2e+1$ questions with boundary $(0,1)$ and $2e+1$ questions with boundary $(n, n+1)$. For otherwise, assume that there exists $i \in \{1,2,\ldots,n-1\}$, such that one of the above $4e+2$ boundaries $(i, i+1)$ is missing. Proceeding as in the proof of the previous theorem, it is possible to define an answering strategy for the adversary that balances the answers on the two sides of the boundary in so that with the information provided by the answers and given the possible number of lies, it is not possible to discriminate between the case $P = \{i\}$ and the case $P = \{i, i+1\}$, or between the case $P = \{i+1\}$ and the case $P = \{i, i+1\}$. Alternatively, if some of the above boundaries $(0,1)$ (resp. $(n, n+1)$) are missing, the adversary can answer in such a way that it is not possible to discriminate between the case $P = \emptyset$ and $P = \{1\}$ (resp. $P = \{n\}$).

4 Bounds for Two-Stage Strategies with One Positive

The aim of this section is to prove asymptotically tight upper and lower bounds on the query number of 2-stage interval group testing algorithms when up to one of the answers is a lie. We shall first analyze the case when P contains *at most one positives.*

We start with some notations and facts which will be used for the proof of the lower bound.

Let $\mathcal{Q}$ be a set of interval questions. For any piece π, cut by $\mathcal{Q}$, we denote by $N(\pi)$ the set of query intervals in $\mathcal{Q}$ containing π.

Let $\pi_1, \ldots, \pi_\ell$ be the pieces determined by the intervals of $\mathcal{Q}$. Given the YES set Y, we define the *weight* it assigns to the piece π_i's according to the following scheme:

- A piece gets weight 1/2 if it can contain a positive and there will not be a lie in the next stage.
- A piece gets weight 3/2 if it can contain a positive and there might be still a lie in the next stage.

Here, "can" means that this possibility is consistent with the YES set.

We denote with $w^Y(\mathcal{Q})$ the weighted sum of the lengths of the pieces cut by $\mathcal{Q}$ weighted according to the weighted associated to Y. In formulas, if w_j is the weight given to the piece π_j, we have $w^Y(\mathcal{Q}) = \sum_{j=1}^{\ell} |\pi_j| w_j$.

Assume now that $\mathcal{Q}$ is the set of interval questions asked in the first stage of a two stage group testing algorithm which finds more than one positive. Using Theorems 1 and 2 it follows that if Y is the set of intervals in $\mathcal{Q}$ that answer YES, the number of queries to be asked in the second stage in order to find all the positives is *at least* $w^Y(\mathcal{Q})$. Since each piece π_j that may have a positive must be solved as an independent interval group testing problem with universe of size $|\pi_j|$ at the second stage, and w_j associates the correct lower bound factor given by Theorems 1 and 2 in the case of one error.
In order to prove the promised bound we will show that for each possible set of interval questions $\mathcal{A}_1$ there exists a yes set Y such that $w^Y(\mathcal{A}_1) \geq n/|\mathcal{A}_1|$.

The following proposition allows us to limit the analysis for the lower bound to a subset of all possible first stages.

Proposition 1. *[1] Let $\mathcal{Q}$ be a set of interval questions producing a partition of the search space in which there are pieces a and b such that $N^{\mathcal{Q}}(a) = N^{\mathcal{Q}}(b)$. Then, there exists a set of interval question $\mathcal{Q}'$ of the same cardinality of $\mathcal{Q}$ such that the following two conditions hold: (i) for each two pieces a' and b' in the partition produced by $\mathcal{Q}'$ it holds $N^{\mathcal{Q}'}(a') \neq N^{\mathcal{Q}'}(b')$; (ii) for each* YES *set Y' for $\mathcal{Q}'$ there exists a* YES *set for $\mathcal{Q}$ such that $w^{Y'}(\mathcal{Q}') = w^Y(\mathcal{Q})$.*

After these preliminaries we can start the proof of the lower bound. Let $\mathcal{Q}$ be the set of questions asked in the first stage by a two stage interval group testing algorithm. Let $q = |\mathcal{Q}|$ In virtue of Proposition 1 we can assume that for each two pieces π_1 and π_2 determined by $\mathcal{Q}$ it holds that $N(\pi_1) \neq N(\pi_2)$. We also have that the total number ℓ of pieces is at most $2q$, since the number of pieces covered by query intervals is at most $2q-1$ (by induction) and by Proposition 1, at most one piece π_o is outside all query intervals ($N(\pi_o) = \emptyset$).

The next technical lemma was proved in [1]. It uses an averaging argument to prove the existence of an adversary strategy that can force a certain number of questions in the second stage.

Lemma 1. *Consider a multiset of k (not necessarily distinct!)* YES *sets, and for each $i = 1, 2, \ldots, k$ and $j = 1, 2, \ldots, \ell$, let w_{ij} be the weight of the jth piece*

in the YES *vector associated to the ith* YES *set. If there exist* $r > 0$ *such that for all* $j = 1, 2, \ldots, \ell$, *it holds that* $\sum_{i=1}^{k} w_{ij} \geq r$, *then an adversary can force at least* $\frac{r}{k}n$ *queries in the second stage.*

We adapt the bounds for the 2-stage strategy for 2 positives, given by Cicalese et al. [1], to the case where a 2-stage strategy for at most one positive may contain at most one error.

Lemma 2

$$\mathcal{N}(n, 1, 2, 1) \geq \sqrt{5n} - O(1).$$

Proof. We show that we may achieve $r \geq 2.5$ using at most $2t_1 + 2$ *YES set*'s, where t_1 is the number of queries in the first stage. Then, by Lemma 1, the number o queries we need is at least $\min\left(t_1 + \frac{5n}{4t_1+4}\right) = \sqrt{5n} - O(1)$.

To achieve $r \geq 2.5$, we create a *specific* YES set for each piece defined by the t_1 queries in the first stage. Recall that there are at most $2t_1$ distinct (according to question containment) pieces. This already guarantees $r \geq 1.5$. Moreover, each pair of adjacent pieces fall in one of the following cases, depending on how many queries separate them:

Case 1. Consider the case where two pieces are separated by the boundary of exactly one query. Let $(i, i+1)$ be such boundary. The YES created for the piece containing i assigns weight $1/2$ to the piece containing $i + 1$, since there is the chance that exactly the query having the boundary $(i, i + 1)$ was an error.

Therefore, by symmetry, each piece in a pair of neighbors separated by a single boundary automatically gets an extra weight $\frac{1}{2}$.

Case 2. When the pieces are separated by the boundary $(i, i + 1)$ of exactly two queries, the YES set created for one of them indicates precisely that piece as the one containing a positive. In these cases, we don't get the extra weight of $\frac{1}{2}$ for the neighbor. However, we can use the fact that, since there is no piece between these two boundaries, the number of pieces is at most $2t_1 - 1$, and so is the number of YES sets used so far. Therefore we may create an *unspecific* YES set involving both pieces. This is a YES set that answer yes to all queries including both pieces and answers the two questions with boundaries $(i, i + 1)$ inconsistently, i.e., one indicating the piece containing i and one indicating the piece containing $i + 1$. This gives us the desired extra weight $\frac{1}{2}$ to each piece.

Case 3. Using the same argument as in the previous case, if a pair of pieces is separated by more than 2 boundaries, then the number of pieces is at most $2t_1 - 2$. We may use two of this extra pieces to create a new specific YES set for each piece in the pair. At the end, each of the pieces gets an extra weight of $\frac{3}{2}$.

Therefore, we are able to extend the previously suggested multiset of YES sets in such a way that each piece gets extra weight $\frac{1}{2}$ from each of its neighbors. As a result, all the pieces, but the ones on the extremities, surely have sum of weights at least 2.5. For pieces on the extremities, creating two extra consistent YES sets, one for each, gives desired total weight. At the end, we have a multiset with the desired properties.

Lemma 3

$$\mathcal{N}(n, 1, 2, 1) \leq \sqrt{5.5n}.$$

Proof. We show a 2-stage query scheme which is able to find a positive in a set of n elements using at most $\sqrt{5.5n}$. The first stage consist in queries divided in two groups, as shown in Fig. 2:

Group A: Consists of t_A overlapping queries that divide the set of objects in $2t_A$ pieces of the same size.

Group B: Consists of rt_A overlapping queries, for $0 < r < 1$.

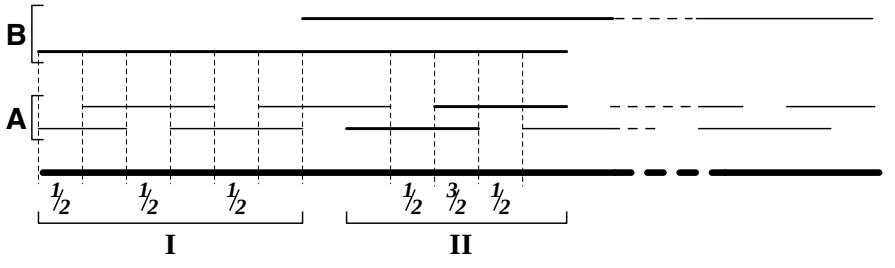


Fig. 2. Query scheme used in the first stage of a 2-stage strategy. The thicker line represents the set of objects, whereas all the other represent the a- and b-queries. In the figure we may see the two patterns that compete for the worst case. Darker lines indicate queries that answer YES.

An inspection of the possible YES sets gives two situations as candidates for the worst case:

I. When a single b-query answers YES correctly, one of the a-queries surely lies. In any case, since at most one error is allowed, the positive must be in one of the single-covered pieces. As a consequence, all such pieces covered by the non overlapping part of the b-query need to be checked in the second stage. Since the non-ovelapping part of the b-querie has size $\frac{n}{2rt_A}$, half of this piece is covered by single a-queries, and an error free strategy may be used in the second stage, the number of queries needed in the following stage is $\frac{n}{8rt_A}$.

II. When two overlapping a-queries answer YES, together with the corresponding b-queries, we must look for a positive in the piece corresponding to the overlapping part as if there was no error. We also need to consider the hypothesis that one of the a-queries gave the wrong answer. Therefore the two pieces corresponding to the non-overlapping parts must also be investigated. In the last case, we may take advantage of the fact that they are only possible in the presence of one error, and use the error-free strategy on the two pieces of size $\frac{n}{2t_A}$. This gives us a total of $\frac{5n}{4t_A}$ questions in the second stage.

The total number of queries used by this strategy is given by
$\min\left(t_A(1+r) + \max\left(\frac{5n}{4t_A}, \frac{n}{8rt_A}\right)\right)$.

By choosing $r = 0.1$, we equalize both worst case candidates and get $\min\left(1.1t_A + \frac{5n}{4t_A}\right) = \sqrt{5.5n}$.

4.1 More Positives

The following theorem summarizes our finding on two stage interval group testing with at most one error in the tests.

Theorem 3

$$\sqrt{6n(p-1)} - O(1) \leq N(n,p,2,1) \leq 2\sqrt{6n(p-1)}.$$

Proof. We start with the lower bound. Assume that the adversary accepts not to lie in the first phase. Moreover, she/he agrees to put the positives into the $p-1$ largest pieces defined by the first stage of queries.

Notice that this information, exchanged between the questioner and the adversary, can only make the situation better for the questioner.

Let q be the number of questions in the first phase. These questions divide the search space into at most $2q+1$ pieces. Hence, the largest $p-1$ of these pieces have total size at least $(p-1)n/(2q+1)$.

Since each of these pieces might contain up to 2 positives, by Theorem 2 the questioner has to ask at least 3 questions per element in each of these pieces.

So we have that the number of questions asked by an algorithm that uses q queries in the first stage is at least $q + 3(p-1)n/(2q+1)$.

Thus, minimizing over all possible values of q we have the desired bound.

In order to prove the upper bound we consider the following strategy, where q is a parameter to be decide later. In the first stage, we divide the search space into q non-overlapping intervals of equal size. We call them segments. Then we ask twice one question coinciding with each segment.

Let $\mathcal{A}$ be the set of segments such that the two corresponding questions are answered YES. Let $\mathcal{B}$ the set of segments whose corresponding questions are answered NO. Finally, let $\mathcal{C}$ the set of segments such that one of the corresponding questions is answered YES and one is answered NO.

Since we are assuming at most one error, trivially, no question is necessary in the second stage in each segment in $\mathcal{B}$.

We also have $|\mathcal{C}| \leq 1$.

We can now have the following cases.

Case 1. $|\mathcal{A}| \leq p-1$, $|\mathcal{C}| = 0$. Then, since each segment π might contain more than 1 positive, and the adversary might still lie, by Theorem 2, $3|\pi|$ questions have to be asked in π in the second stage. Since all segments are of the same size, in total we have $2q + 3|\mathcal{A}|n/q$ questions are asked in this case.

Case 2. $|\mathcal{A}| \leq p-1$, $|\mathcal{C}| = 1$. Again, each segment π might contain more than 1 positive. However, in this case the adversary has clearly already used a lie. Then, by Theorem 2, for each segment $\pi \in \mathcal{A}$, $|\pi|$ questions have to be asked in π in the second stage. Moreover, for the only segment γ in $\mathcal{C}$, either $|\gamma|/2$ or $3|\gamma|/2$

questions have to be asked, according as $\mathcal{A}$ contains $p-1$ or less segments. In fact, in the first case, γ can contain at most one positive, and Theorem 1 applies. Whilst in the second case, γ might contain more than one positive and then Theorem 1 applies. Since all segments are of the same size, in total we have $2q + 3(p-1)n/2q + n/2q$ questions in the first case and $2q + 3(|\mathcal{A}| + 1)n/2q$ in the second case ($|\mathcal{A}| \leq p-2$).

It is not hard to see that the worst situation for the questioner is given by *Case 1* with $|\mathcal{A}| = p-1$.

Thus, the above strategy uses in total at most $2q + 3(p-1)n/q$ questions. Minimizing with respect to q we have the desired bound.

References

1. Cicalese, F., Damaschke, P., Vaccaro, U.: Optimal group testing strategies with interval queries and their application to splice site detection. Int. Journal of Bioinformatics Research and Applications
2. Cicalese, F., Damaschke, P., Tansini, L., Werth, S.: Overlaps Help: Improved Bounds for Group Testing with Interval Queries. Discrete Applied Mathematics (to appear)
3. Cormode, G., Muthukrishnan, S.: What's hot and what's not: Tracking most frequent items dynamically. In: ACM Principles of Database Systems (2003)
4. Cox, L.A., Sun, X., Qiu, Y.: Optimal and Heuristic Search for a Hidden Object in one Dimension. In: Proc. of IEEE Conf. on System, Man, and Cybernetics, pp. 1252–1256 (1994)
5. Du, D.Z., Hwang, F.K.: Combinatorial Group Testing and its Applications. World Scientific, Singapore (2000)
6. Farach, M., Kannan, S., Knill, E.H., Muthukrishnan, S.: Group testing with sequences in experimental molecular biology. In: Carpentieri, B., De Santis, A., Vaccaro, U., Storer, J. (eds.) Proc. of Compression and Complexity of Sequences 1997, pp. 357–367. IEEE CS Press, Los Alamitos (1997)
7. Knill, E.: Lower Bounds for Identifying Subset Members with Subset Queries. In: Proceedings of Symposium on Discrete Algorithms 1995 (SODA 1995), pp. 369–377 (1995)
8. Gelfand, M., Mironov, A., Pevzner, P.A., Roytberg, M., Sze, S.H.: PROCRUSTES: Similarity-based gene recognition via spliced alignment, `http://www-hto.usc.edu/software/procrustes/`
9. Hong, E.H., Ladner, R.E.: Group testing for image compression. IEEE Transactions on Image Processing 11(8), 901–911 (2002)
10. Hong, Y.W., Scaglione, A.: On multiple access for distributed dependent sources: A content-based group testing approach. In: IEEE Information Theory Workshop ITW (2004)
11. Hong, Y.W., Scaglione, A.: Group testing for sensor networks: the value of asking the right answers. In: Asilomar Conference (2004)
12. Hong, Y.W., Scaglione, A.: Generalized group testing for retrieving distributed information. In: ICASSP (2005)
13. Karp, R.: ISIT 1998 Plenary Lecture Report: Variations on the theme of Twenty Questions. IEEE Information Theory Society Newsletter 49(1) (1999)

14. Ngo, H.Q., Du, D.Z.: A survey on combinatorial group testing algorithms with applications to DNA library screening. Discrete Mathematical Problems with Medical Applications. DIMACS Ser. Discrete Math. Theoret. Comput. Sci., Amer. Math. Soc. 55, 171–182 (2000)
15. Pevzner, P.A.: Computational Molecular Biology, An Algorithmic Approach. MIT Press, Cambridge (2000)
16. Sobel, M., Groll, P.A.: Group testing to eliminate efficiently all defectives in a binomial sample. The Bell Systems Technical Journal 38, 1179–1253 (1959)
17. Wolf, J.: Born again group testing: Multiaccess communications. IEEE Trans. Information Theory IT-31, 185–191 (1985)
18. Xu, G., Sze, S.H., Liu, C.P., Pevzner, P.A., Arnheim, N.: Gene hunting without sequencing genomic clones: Finding exon boundaries in cDNAs. Genomics 47, 171–179 (1998)

Approximate String Matching with Swap and Mismatch

Ohad Lipsky[1], Benny Porat[1], Elly Porat[2], B. Riva Shalom[1], and Asaf Tzur[1]

[1] Department of Computer Science, Bar-Ilan University, Ramat-Gan 52900, Israel
{lipsky,bennyporat,riva.shalom}@gmail.com,
zurasa@cs.biu.ac.il
[2] Department of Computer Science, Bar-Ilan University,
Ramat-Gan 52900, Israel, +972 3 531-7620,
IDF and Google incorporation
porately@gmail.com

Abstract. Finding the similarity between two sequences is a major problem in computer science. It is motivated by many issues from computational biology as well as from information retrieval and image processing. These fields take into account possible corruptions of the data caused by genome rearrangements, typing mistakes, and more. Therefore, many applications do not require merely complete resemblance of the sequences, but rather an approximated matching. We consider mismatches and swaps as natural mistakes which are allowed in a meagre number. The edit distance problem with swap and mismatch operations was discussed by Amir et. al. [3]. They solved the problem in $O(n\sqrt{m}\log m)$ time. From then on the problem of string matching with at most k swaps and mismatches errors was open.

In this paper we present an algorithm that finds all locations where the pattern has at most k mismatch and swap errors in time $O(n\sqrt{k\log m})$.

1 Introduction

Before describing the motivation of the problem, we share with the reader the importance of solving the Approximate String Matching with Swap and Mismatch errors, being an open problem for over three years, worked on by some researchers groups to no avail. Only recently, a randomized solution was suggested [7].

In the last few decades various scientific and business applications requested solutions for the Approximate String Matching Problem. In approximate matching, one defines a distance metric between the objects (e.g. strings, matrices) and seeks all text location where the pattern matches the text by a pre-specified "small" distance. The necessity mainly derives from information retrieval, image processing and computational biology. In the latter, for example, we know that during the course of evolution, speciation results in the divergence of genomes that initially have the same gene order and content. If there is no selective pressure, successive rearrangements that are common in prokaryotic genomes will

T. Tokuyama (Ed.): ISAAC 2007, LNCS 4835, pp. 869–880, 2007.

eventually lead to a randomized gene order. Therefore the presence of a region of conserved gene order is a source of evidence of some non-random signal that allows, for example, the prediction of groups of functionally associated genes [13]. Nevertheless, as rearrangements do occur, the approximate matching is sought after, as well as the exact matching. Similarly, detecting that a certain protein is in proximity to a known one, may yield functional similarity between them, giving biologists a lead for their research.

Definition 1. *Let S and Q be two strings over alphabet Σ and let E be a set of edit operations. Then the* edit distance*(S, Q) with respect to E is the minimum number d, such that exists a sequence of d edit operations $\in E$ for which $e_d(e_{d-1}(...e_1(Q)...)) = S$.*

Definition 2. *Given a string $S = s_0...s_{n-1}$ over alphabet Σ and $\sigma \in \Sigma$,* ***swap*** *$((i)(S)) = s_0...s_{i-1}s_{i+1}s_is_{i+2}...s_{n-1}$.* ***mis****$((i, \sigma)(S)) = s_0...s_{i-1}\sigma s_{i+1}...s_{n-1}$.*

The earliest and best known distance functions are Levenshtein's *edit distance* [10] and the *Hamming distance*, considering merely mismatches. Let n be the text length and m the pattern length. Lowrance and Wagner [11,12] proposed an $O(nm)$ dynamic programming algorithm for the extended edit distance problem. In [9] the first $O(kn)$ algorithm was given for the edit distance with only k allowed mismatches. Cole and Hariharan [6] presented an $O(nk^4/m + n + m)$ algorithm for this problem. To this day, however, there is no known algorithm that solves the general case of the extended edit distance problem, where the edit operations are: insertion, deletion, mismatch, and swap, in time $o(nm)$.

Since the upper bound for the edit distance seems very tough to break, attempts were made to consider the edit operations separately. If only mismatches are counted for the distance metric, we get the *Hamming distance*, which defines the *string matching with mismatches* problem. A great amount of work was done on finding efficient algorithms for string matching with mismatches among which are [9,5]. The most efficient deterministic worst-case algorithm for finding the Hamming distance of the pattern at every text location runs in time $O(n\sqrt{m \log m})$. Isolating the swap edit operation yielded even better results [2,4], with a worst-case running time of $O(n \log m \log \sigma)$.

Amir, Eisenberg and Porat [3] faced the challenge of integration of the above two results, providing an efficient algorithm for edit distance with mismatch and swap. Integration can prove to be tricky, the problem of indexing with don't cares can serves as an example for that.

In fact, sometimes the integration of two efficiently solvable operations ends up intractable. For example, Wagner [12] proves that edit distance with the *two* operations: *insertion* and *swap* in $\mathcal{NP}$-hard, while each one separately can be solved in polynomial time and the general edit distance – consisting of the *four* operations *insertion*, *deletion*, *mismatch* and *swap* – is also polynomially solvable.

In this paper we discuss the problem of approximated string matching, with only swap and mismatch operations allowed, as formally defined in Definition 3.

Definition 3. *The Approximated String Matching with Swaps and Mismatches problem (ASMSM):*

Input: *A text* $T = t_0...t_{n-1}$ *and a pattern* $P = p_0...p_{m-1}$*, over alphabet* Σ*, a constant* k *and a set of editing operations* $E = \{swap(i), mis(i, \sigma)\}$.

Output: All text locations i*, for which the edit distance, with respect to* E*, between* P *and the substring* $t_i..t_{i+m-1}$ *is* $\leq k$*, with the restriction that each character can participate in no more than one swap.*

We present an $O(n\sqrt{k \log m})$ time algorithm for a constant alphabet, using counting techniques, convolutions and other combinatorial methods.

The paper is organized as follows: In the next section we give some related previous works. In section 3 we describe our algorithm, parted to three cases. Section 4 describes an efficient method for detecting swaps. Section 5 concludes the paper.

2 Previous Work

The ASMSM problem integrates challenges of both approximating matching and that of confronting the swap and mismatch operations. We therefore briefly scan results regarding both questions.

Landau and Vishkin [9] solved the approximated matching, allowing k mismatches, in $O(nk)$ time, for a text of length n. They introduced a method of using suffix trees and Lowest Common Ancestor queries in order to allow constant-time jumps over equal substrings in the text and the pattern.

Their solution can be easily adjusted to solve our problem, allowing swaps as well. Start at each text location, pass over the longest equal substring, till a mismatch is reached, consuming constant time (using a LCA query). After the next 'jump' check wether the last mismatches are adjacent, and if so, whether a single swap can replace the two mismatches. In case a swap was found, the number of corrections activated so far is updated. If a location has more than k errors, we stop. Thus, verification of every location takes time of $O(2k)$. Hence, the total time required is $O(nk)$, a bound which we intend to improve in this paper.

Amir, Lewenstein and Porat [5] solved the problem of approximated pattern matching, where k mismatches are allowed per location . They introduced a counting technique reducing the number of text locations needed to be checked using filtering. Their algorithm requires $O(n\sqrt{k \log k})$ time. Their solution does not allow us direct access to the text locations during its process, hence, it cannot efficiently solve the ASMSM problem. Nevertheless, we will follow some ideas from their algorithm, but instead of symbols we will refer them to segments of the text and the pattern.

Amir et. al. [1] showed that if the swap operation is isolated as the only edit distance operation allowed, the approximated string matching problem can be solved in time $O(n\sqrt{m \log m})$, where n, m stands for the length of the text, pattern, correspondingly.

The first integration of the swap and mismatch as the set of legitimate operations of the edit distance was suggested by Amir Eisenberg and Porat [3]. They considered the case of binary alphabet and developed an algorithm using novel cases of overlap matching and convolutions, consuming $O(n \log m)$ time, where n, m stands for the length of the text and pattern, correspondingly. For general alphabet they reduced part of the problem to binary alphabet, yielding a total solution in $O(n\sqrt{m} \log m)$ time. Obviously their algorithm solves our problem. Using their application, the time required for solving our problem is $O(n\sqrt{m} \log m)$, which we wish to ameliorate.

Lately, Porat et. al. [7] solved the Approximate Swap and Mismatch Edit Distance using a randomized algorithm.

3 The Algorithm

The algorithm we suggest for the Approximate String Matching with Swaps and Mismatches problem uses several filtering yielding possible candidates and then verifying these text locations.

As a first step, we partition the text to pieces of length $2m$ starting from the first text symbol. In order to avoid neglecting pattern appearances that are divided between two adjacent text pieces, we perform yet another partitioning starting from index m of the text. As a consequence we have the following texts: $[T_0, T_{2m-1}]$, $[T_2m, T_{4m-1}]$, ... and $[T_m, T_{3m-1}]$, $[T_3m, T_{5m-1}]$, ... The creation of the text pieces is done in linear time and space. Throughout the algorithm we consider these text pieces as texts of size $2m$. The results returned by the algorithm are later normalized by the start index of the text piece, in the input text of size n, to form the output.

Consider the combination of our edit operations, swap and mismatch. It is clear that every swap error can be viewed as two mismatch errors, yielding the following observation.

Observation 1. *Every text location with at most k swap and mismatch errors can not have more than $2k$ mismatch errors.*

As a primary filter we use the algorithm of Amir Lewenstien and Porat [5] for string matching with a constant number of mismatches. Due to Observation 1 we apply their algorithm with $2k$ allowed mismatches. This procedure requires $O(m\sqrt{k \log k})$ time. To every text location we attach a flag, which is set, only if its corresponding location begins an occurrence of the pattern with at most $2k$ mismatches. From now on, we process the whole text but consider merely results associated with possible candidates.

The next step is to perform a sort of relabelling over the text and pattern, by converting the foundation stones of the sequences from single symbols to segments.

Definition 4. An alternating segment *of a string S over alphabet Σ, is a substring alternating between $\sigma, \rho \in \Sigma$.* A maximal alternating segment*, or a* segment *for short, is an alternating segment that cannot be expanded to either side.*

In other words, if S' is a segment $\sigma\rho\sigma\ldots$, then the character to the left of S', cannot be ρ and the one to the right of S' is distinguished from the symbol before the last symbol of S'. A segment is therefore uniquely defined by its first and second characters and by its length. To each segment we attach its appearance index in the sequence. When partitioning a string S into segments, we start a segment at the last symbol of the preceding one, implying a single character overlap between two consecutive segments. An example for segmentation is: $S = ababcbbacacbabab$, then the segments are: $abab[a,b,4](1)$, $bc[b,c,3](4)$, $b[b,b,1](6)$, $ba[b,a,2](7)$, $ac[a,c,4](8)$, $cb[c,b,2](11)$, $babab[b,a,5](12)$.

Avoiding the segments overlap, we can err counting correctly the swaps mistakes between pattern segments and text segments. For example, suppose $P = ababacac$, its non-overlapping segment are $Sp_1 = ababa$ and $Sp_2 = cac$. If the text contains $babacaca$, we will count for Sp_1 two swaps and a mismatch, and for Sp_2 a single swap and a single mismatch, summing up to 5 mistakes, while the correct number of mistakes is 4 swaps. The partitioning including border symbols overlaps prevents such pithalls. Nevertheless we must ensure that overlapping symbol will not participate in two swap operations.

Claim. For every two consecutive segments s_1, s_2, either one of the last symbol of s_1 and the first symbol of s_2 takes part in a swap correction or neither of them does.

Proof: Let $s_1 = ...ab$. Since s_2 is adjacent to s_1, it must begin in b due to the border symbol overlap. In addition, we know that the consecutive symbol is not a, otherwise contradicting the maximality of s_1, therefore $s_2 = bc...$ Suppose the last ab of s_1 are swapped, hence the text segment St compared to s_1 is $St = ..baba...$ If s_2 is also compared to St right after s_1, it will confront ba aligned to $s_2 = bc..$, yielding no need for swapping. Nevertheless, if St's end is aligned to the end of s_1, even though s_2 will be aligned to a different segment, it is the consecutive segment of St, therefore starting in a and continuing in a symbol distinct from b, here again implying no profit from a swap operation. When s_2 takes part in a swap the proof is symmetric. ∎

After executing these primary procedures of constructing a new text and pattern ST, and SP by parting them to segments and replacing their symbols by the appropriate segments, we are ready for efficiently solving the ASMSM problem. For this purpose we consider three cases of pattern instances, for each we suggest appropriate methods.

3.1 Pattern with Many Different Segments

In this subsection we deal with patterns having more than $3k$ distinct segments. Where a distinct segment refer to a segment different from all other segments, by at least one of its properties, starting letter, second letter and its length, avoiding duplicity. For this case we suggest finding all pattern occurrences in the text with no more than k swaps and mismatches in linear time. This will be done using a counting filter based on the following observation, and a verification.

Observation 2. *A pattern segment that does not match exactly an identical text segment produces at least one mismatch error.*

Proof : Suppose a text segment t is aligned with segment p of the pattern. By definition, if the segments do not match in the overlap, there are mismatches. Suppose they do match in the overlap area, but one of them, say t, ends after the other, i.e. p ends at location i and t ends at location $i+j$, for $j > 0$. As a consequence, there is a mismatch at the location $i+1$, otherwise, the symbol at location $i+1$ of the pattern was the same as that of t and the segment p could have been continued, contradicting the correctness of the segmentation. The case where one segment starts before the other is symmetric. ∎

We would like to count, for every text location l, how many pattern segments are identical to the text segments included in $T[l...l+m-1]$. Let $\{Sp_1, , Sp_{3k}\}$ be the first $3k$ different pattern segments. We select for each Sp_j its first occurrence in the pattern starting at index i_j. Now, for every text location l, if l is a start of a text segment St and there exists a pattern segment Sp_j that is **identical** to St, then mark, increment by one the counter related to text location $l - i_j$.

Consequentially to Observation 1 and to our marking $3k$ pattern segment occurrences in text segments, a possible matching must be marked by at least k pattern segments out of the $3k$, allowing at most $2k$ mismatches due to $2k$ pattern segments that did not match. Thus, locations marked by less than k pattern segments are discarded.

Observation 3. *At the conclusion of the marking stage there are at most* $O(2m/k)$ *candidate locations.*

Proof: The algorithm performs at most one marking per text location, the total number of marks cannot exceed $2m$. Every location that was not discarded has at least k marks, there could be no more than $2m/k$ such locations. ∎

For verifying the $2m/k$ new candidates we use the Landau and Vishkin [9] method of moving from one mismatch to the next one, till the $(k+1)$th mismatch is reached. As we go over the mismatches we consider consecutive ones and check, with no additional complexity, whether they can be corrected by a single swap and if so we increase the number of yet allowed mistakes by one. This verification requires $O(2k)$ time per candidate location. As a consequence the time required for verifying all candidates is $O(km/k) = O(m)$.

Lemma 1. *The ASMSM problem for patterns with more than* $3k$ *distinct segments is linearly solvable.*

3.2 Pattern with Frequent Segments

In this subsection we deal with patterns having fewer distinct segments that appear in higher frequency than in the previous case. Formally, in this case there are at least $\sqrt{k/\log m}$ different frequent segments each occurring at least $3\sqrt{k \log m}$ times in the pattern.

Note that in order to apply the marking and counting technique, we need to choose for marking purposes 3k pattern segments, a condition which is fulfilled by selecting the first $3\sqrt{k \log m}$ appearances of each of the different frequent segments.

For each frequent pattern segment Sp we will construct a list of indices j, starting locations of the $3\sqrt{k \log m}$ first occurrences of Sp in the pattern. We then go over the text and for each text location l that is a start of a segment St in the text, identical to a segment Sp, we will mark locations $l - j$ for each j in the list. Since we choose $3k$ segments' occurrences in total, we will need $O(k)$ additional space for the lists.

After the marking step, we discard every text location that is marked with less than k marks.

Observation 4. *At the conclusion of the marking stage there are at most* $3m \cdot \sqrt{\log m/k}$ *candidate locations.*

Proof: Similar to the proof of Observation 3, the algorithm can perform $3\sqrt{k \log m}$ marking per text location, the total number of marks cannot exceed $3m\sqrt{k \log m}$. Every location that was not discarded has at least k marks, hence, there could be no more than $3m\sqrt{k \log m}/k = 3m\sqrt{\log m/k}$ such locations.

For verifying the candidates we use, here again, the Landau and Vishkin [9] method, as was previously described. Since there are at most $3m\sqrt{\log m/k}$ candidate locations, and the verification time for each candidate is $O(k)$, the total verification time is $O(m\sqrt{k \log m})$.

Lemma 2. *The case of patterns with* $\sqrt{k/\log m}$ *frequent segments is solvable in time* $O(m\sqrt{k \log m})$.

If there are less than $\sqrt{k/\log m}$ frequent segments, we will choose $3\sqrt{k}$ occurrences of every frequent segment, and then we will pick unfrequent segments and their occurrences. If we manage to gather $3k$ segments' occurrences altogether, then the necessary condition is satisfied and we obtain $O(m\sqrt{k \log m})$ time algorithm once again using counting arguments.

3.3 Few Segments, Fewer Frequents

The last case considers the case where there are less than $\sqrt{k/\log m}$ frequent pattern segments, and there are no $3k$ segments appearances in the pattern required for the counting filtering.

For the current case we suggest to compute the swaps caused by all possible alignments of the pattern on the text. Recall we have calculated, at the beginning of the algorithm, all text locations matching the pattern with at most $2k$ mismatches. If we can detect the number of swaps, subtracting the latter from the former will give us the number of required swaps and mismatches in matching the pattern to each of the candidate text location. Locations that do not exceed k errors are reported as output.

For text T of length $2m$, we consider all its first $m+1$ substrings, possible full overlaps with the pattern, and call this substring a *text section.*

We differ between frequency of segments as follows: A text section segment is called *frequent* if it appears at least $5\sqrt{klogm}$ times in the text section. We determine that a text segment is called *frequent* if it appears at least $10\sqrt{klogm}$ times in the text (of length 2m). Recall a pattern segment is called *frequent* if it appears at least $3\sqrt{klogm}$ times in the pattern. Note that these rather high constants were selected for the ease of the reader, in practice they can be substantially reduced.

For each text section we count the number of its frequent segments. Observe, that as a substring differs from its proceeding one by omitting its first segment and by the adding of a single segment at its end, using a sliding window, this process is done in linear time, for all text sections.

Lemma 3. *In case the pattern contains less than* $\sqrt{k/\log m}$ *frequent segments and a text section consists of at least* $2\sqrt{k/\log m}$ *frequent segments, this section cannot be matched to the pattern under the problem restrictions.*

Proof: Suppose, in the worst case, all frequent segments of P are contained in the text frequent segments set, hence, there are at least $\sqrt{k/\log m}$ frequent segments of the text section which do not frequently occur in P. Even if we claim that all these segments do appear in P almost frequently, up to $3\sqrt{klogm}-1$ times, these $\sqrt{k/\log m}$ segments still have additional $2\sqrt{klogm}+1$ appearances in the text segment that have no correspondence in P , due to the frequency definitions. Hence, we get at least than $\sqrt{k/\log m}\cdot 2\sqrt{klogm} = 2k$ mismatches, preventing an approximate matching between this text section and the pattern. ∎

As this subsection considers merely patterns with less than $\sqrt{k/\log m}$ frequent segments, due to Lemma 3, we can rule out text locations by their number of frequent segments information. We go over our text, first from right to left and then from left to right and seek the first location possessing less than $2\sqrt{k/\log m}$ frequent text segments. These text locations, bounds the area in the text of possible candidates.

Lemma 4. *The bounded area contains* $c\sqrt{k/\log m}$ *frequent segments, c a small constant.*

Proof: The bounds of the bounded area are locations that the text sections starting there have less than $2\sqrt{k/\log m}$ frequent text section segments. Even if these locations are far enough, where their frequent text section segments are not overlapping, and the frequent segments are identical, implying their frequency is multiplied by 2, we get only $2\sqrt{k/\log m}$ frequent text segments.

Moreover, though there may be more frequent text section segments, because a non frequent text section segment now appears twice the time and can become frequent, nevertheless, it cannot attain the frequency demand in the total text. ∎

For the frequent segments we use the Amir, Eisenberg, Porat [3] algorithm for calculating {swap, mismatch} edit distance of sequences with binary alphabets,

requiring $O(n \log m)$ time, where the text is of size n and the pattern of size m. Their algorithm computes the number of real mismatches due to certain pattern and text segments alignments, yet their output can easily be translated to swaps, (by executing an additional convolution of the appropriate sequences, and subtracting the output from it and dividing by two). Due to Lemma 4 and the current case of few frequent pattern segments, there are merely $c\sqrt{k/\log m}$ frequent segments from the pattern and text, to consider. In order to avoid duplicate counting, we operate the algorithm for every frequent pattern segment with all segments of the text, and for frequent text segments with merely unfrequent pattern segments. Therefore, the time consumed by calculating their swaps contribution is $O(m\sqrt{k/\log m}\log m) = O(m\sqrt{k \log m})$. ∎

Lemma 5. *Counting the number of swaps induced by frequent segments can be done in $O(m\sqrt{k \log m})$ time.*

Having calculated the number of the swaps caused by the frequent segments from both pattern and text, we are left to count swaps caused by non frequent pattern segments (appearing at most $3\sqrt{k \log m}$ times in the pattern), that come across a non frequent text segment (appearing at most $10\sqrt{k \log m}$).

In Section 4 we suggest a procedure, that given a pattern and text segments Sp and St, marks the number of swaps due to comparing the segments in constant time, enabling to retrieve by two passes over the text the total number of swaps due to all Sps. First we consider matching of Sp of length i with all text segments of length greater than or equal to i and then the other way around.

Lemma 6. *Marking the swaps caused by alignments of pattern segments and texts segments of greater or equal size can be done in $O(m\sqrt{klogm})$ time.*

Proof: For a text segment St of length $|St|$, the number of different pattern segments of length $|St|$ or less is bounded by $2|St|\Sigma^2$ due to $|St|$ possible lengths, Σ^2 options for alphabet and two options for the first character. Each of the pattern segments can appear at most $3\sqrt{k \log m}$ times and every appearance implies a $O(1)$ time for marking the swaps by Lemma 8. We get that a single text segment St requires $6|St|\sqrt{k \log m}\Sigma^2$ time for the marking. Since we deal with small sizes alphabets we say the marking per a text segment is done in $O(|St|\sqrt{k \log m})$. The time required for all text segments matched to shorter pattern segments is $\sum_{St \in T} O(|St|\sqrt{k \log m})$. All text segments compose the text itself, so we get a time of $O(m\sqrt{k \log m})$. ∎

Now we need to compare Sp of length i with all text segments of length less than i. We apply the marking procedure here again, but count the number of operations from the pattern segment point of view, implying it can be matched to shorter text segments, whose number is bounded by $2(|Sp| - 1)\Sigma^2$. Recall that Sp's appearances in the current case, can not exceed $3\sqrt{k \log m}$ in the pattern. Therefore, The time required for comparing Sp to shorter text segments is $6\Sigma^2(|Sp| - 1)\sqrt{k \log m} = O(|Sp|\sqrt{k \log m})$. Considering all pattern segments, the time is $\sum_{Sp \in P} O(|Sp|\sqrt{k \log m})$ and since all Sps construct P, the overall time is $O(m\sqrt{k \log m})$.

Lemma 7. *The time required for solving the ASMSM problem for the third case of patterns is* $O(m\sqrt{k \log m})$.

Proof: Due to Lemmas 5, 6.

Theorem 1. *The Approximate String Matching with swaps and mismatches problem is solvable in* $O(n\sqrt{k \log m})$ *time.*

Proof: Due to Lemmas 1, 2, 7 we have the problem solved for $2m$ sized texts in $O(m\sqrt{k \log m})$. We perform the algorithm for each of the $2n/m$ text pieces, yielding the required time. ∎

Suppose $k = O(m)$ then, $\sqrt{k} = O(\sqrt{m})$. For small value k there is an additional algorithm, solving the ASMSM problem in $O(m\sqrt{k \log k})$ time, to be published.

4 Detecting Swaps

The problem we solve in this section is, efficiently marking the number of swaps between two segments St and Sp.

Observation 5. *Detecting swaps between two segments, only segments sharing the same alphabet need to be considered.*

Proof: For two segments, there are three possible relations concerning their alphabets: They can be distinct, share a single character or they can be identical. For the first case, no swap can obtain a match between the segments. If the segments share a single symbol, for example $Sp = abab$, $St = cbcbc$ then activating a swap operations over Sp, will not reduce the edit distance between the segments, as each pair of symbols will require a swap and a replacement corrections instead of two replacements, so all mismatches are real. When the alphabets of the segments St, Sp are identical, swaps do occur. The first symbol of St and the last of Sp determine the number of swaps required for matching the segments. ∎

For simplicity, we suppose hereafter that the pattern segment Sp is the first segment of the pattern, therefore, when aligning P to index l of T, all mismatches found are due to index l of T. If this is not the situation, and Sp begins at index i of the pattern, the number of mismatches calculated should be written associated to location $l - i$ of T.

Consider, for example, $St = abababa$ and $Sp = baba$ some possible alignments of these segments are depicted in Figure 1. Beneath a text location l we write the number of swaps between St and Sp due to placing Sp on location l of T.

Marking each text location with the number of swaps can be easily done in linear time. However, we try to reduce the time complexity, and suggest writing on a new clear array named *change* merely changes in the overlap situations. When Sp start overlapping St on $T[j]$, the overlap contains a single symbol. Hereafter, the overlap increases until it reaches its pick. We perform $change[j] = change[j] + 1$ meaning that from now on the number of swaps increases by one for every alternate offset. Having reached the largest possible overlap, starting at index j',

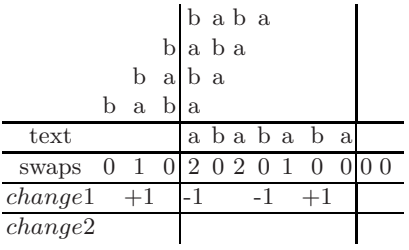


Fig. 1.

the number of swaps remains the same, so we want to stop the incrementing by $change[j'+2] = change[j'+2] - 1$.
The next change occurs when the overlap decreases from index j'', we need a further decrementing and update $change[j''] = change[j''] - 1$. Nevertheless, if the length of the overlap reaches its pick, at index j' and start decreasing in the following location, we will then need to decrease 2 from $change[j'+2]$, to counteract the first +1 as well as start decreasing by one.
When the overlap ends at index j''', we want to avoid a negative number of mismatches formed by the continuation of the -1 usage, by performing $change[j'''] = change[j''']+1$ and stabilizing the zero number of swaps due to the current Sp, St.

We save two *change* arrays, one for odd offsets and the other for the evens. In the example of Figure 1, supposing the first text location is 0, the odd offsets are the swapping locations, so we use $change1$. The arrays are updated for every relevant Sp, St. After all segments of the same alphabets were compared, we calculate, via a single pass over the *change* arrays, the number of swaps per text location which will be saved in array *swap*. We also use a counter dif standing for the difference between the current *swap* entry and the preceding one. The computation is done consecutively by the following assignments:

$$if\ i\ is\ odd \quad dif1 = dif1 + change1[i], \quad swap1[i] = swap1[i-2] + dif1. \quad (1)$$

$$if\ i\ is\ even \quad dif2 = dif2 + change2[i], \quad swap2[i] = swap2[i-2] + dif2. \quad (2)$$

Accumulating the number of swaps due to all segments is done by summing $swap1$ and $swap2$.

Note, that there may be several swapped overlaps in a certain offset l of P on T due to different pattern segments, yet they will be noted when compared, and the appropriate adding will be done on $change[l]$. As a consequence, in the final pass the appropriate values will be added to $swap[l]$, capturing all swaps altogether (this is the reason for adding values to *change* instead of assigning it the value).

In case the last symbol of Sp is not the same as the first of St, we will activate the same operations on $change2$ at the appropriate locations. Determining the text locations in which *change* should be updated, and the values added there, can be easily done in constant time. The details will appear in the full version of the paper.

Lemma 8. *Marking the number of swaps between a pattern segment and a text segment requires constant time.*

5 Conclusions

The main contribution of this paper is presenting a simple yet efficient algorithm for the important problem of approximated matching with swap and mismatch errors. We have used counting and convolution techniques, adjusting them to this problem unique requirements, as well as other combinatorial methods. Other questions of finding the distance or approximated matching between two sequences with regard to other sets of editing operations are still open.

References

1. Amir, A., Aumann, Y., Landau, G.M., Lewenstein, M., Lewenstein, N.: Pattern Matching with Swaps. In: Proc.38th IEEE FOCS, pp. 144–153 (1997)
2. Amir, A., Cole, R., Hariharan, R., Lewenstein, M., Porat, E.: Overlap matching. Inf. Comput. 181(1), 57–74 (2003)
3. Amir, A., Eisenberg, E., Porat, E.: Swap and Mismatch Edit Distance. In: Albers, S., Radzik, T. (eds.) ESA 2004. LNCS, vol. 3221, pp. 16–27. Springer, Heidelberg (2004)
4. Amir, A., Lewenstein, M., Porat, E.: Approximate swapped matching. Inf. Process. Lett. 83(1), 33–39 (2002)
5. Amir, A., Lewenstein, M., Porat, E.: Faster algorithms for string matching with k mismatches. J. Algorithms 50(2), 257–275 (2004)
6. Cole, R., Hariharan, R.: Approximate string matching: A faster simpler algorithm. In: Proc. 9th ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 463–472 (1998)
7. Dombb, Y., Lipsky, O., Porat, B., Porat, E., Zur, A.: Approximate Swap and Mismatch Edit Distance. In: the 14th String Processing and Information Retrievial Symposion (SPIRE) (to appear, 2007)
8. Karloff, H.: Fast algorithms for approximately counting mismatches. Information Processing Letters 48(2), 53–60 (1993)
9. Landau, G.M., Vishkin, U.: Efficient String Matching with k Mismatches. Theoretical Computer Science 43, 239–249 (1986)
10. Levenshtein, V.I.: Binary codes capable of correcting, deletions, insertions and reversals. Soviet Phys. Dokl. 10, 707–710 (1966)
11. Lowrance, R., Wagner, R.A.: An extension of the string-to-string correction problem. J. of the ACM, 177–183 (1975)
12. Wagner, R.A.: On the complexity of the extended string-to-string correction problem. In: Proc. 7th ACM STOC, pp. 218–223 (1975)
13. Yanai, I., DeLisi, C.: The society of genes: networks of functionallinks between genes from comparative genomics. Genome Biol. 3(64), 1–12 (2002)

Minimum Fill-In and Treewidth of Split+ke and Split+kv Graphs*

Federico Mancini

Department of Informatics, University of Bergen,
N-5020 Bergen, Norway
federico@ii.uib.no

Abstract. In this paper we investigate how graph problems that are NP-hard in general, but polynomially solvable on split graphs, behave on input graphs that are close to being split. For this purpose we define split+ke and split+kv graphs to be the graphs that can be made split by removing at most k edges and at most k vertices, respectively. We show that problems like treewidth and minimum fill-in are fixed parameter tractable with parameter k on split+ke graphs. Along with positive results of fixed parameter tractability of several problems on split+ke and split+kv graphs, we also show a surprising hardness result. We prove that computing the minimum fill-in of split+kv graphs is NP-hard even for $k = 1$. This implies that also minimum fill-in of chordal+kv graphs is NP-hard for every k. In contrast, we show that the treewidth of split+$1v$ graphs can be computed in polynomial time. This gives probably the first graph class for which the treewidth and the minimum fill-in problems have different computational complexity.

1 Introduction

Many NP-hard graph problems become polynomially solvable when restricted to specific graph classes. Let $\mathcal{C}$ be such a class. A natural question is whether a problem that is tractable on $\mathcal{C}$ remains tractable when we consider a graph class that is close to $\mathcal{C}$. For example a class where every graph can be made into a graph of $\mathcal{C}$ altering only few edges or vertices. This is a common situation in practical applications, where the input can be affected by errors or incomplete information. For this purpose we define the graph classes $\mathcal{C}+ke$, $\mathcal{C}-ke$ or $\mathcal{C}+kv$ to be the class of graphs that can be obtained by, respectively, adding at most k edges, removing at most k edges or adding at most k vertices to the graphs in $\mathcal{C}$. Notice that when dealing with hereditary properties, it does not make sense to remove vertices. We will often refer to these classes of graphs as *parametrized graph classes*.

Recently several authors studied the complexity of hard problems on such graph classes from a parametrized point of view [4,5,29,22,11]. In particular in [11] the

* This work is supported by the Research Council of Norway through grant 166429/V30.

T. Tokuyama (Ed.): ISAAC 2007, LNCS 4835, pp. 881–892, 2007.

whole idea is generalized and the parameter k is considered as a measurement of the "distance from triviality". In other words k is used to measure how the complexity of a problem changes as we get further from a trivial solution. For a given problem and a graph class on which it has a polynomial time solution, the main questions we can ask about the corresponding parametrized classes are: Does the problem remain polynomial time solvable up to some k and become NP-hard for $k+1$ or does it remain polynomial for each fixed k? And in the last case, how does it behave from a parametrized complexity point of view? Is it FPT or W-hard? A problem with parameter k is fixed parameter tractable (FPT) or *uniformly polynomial*, if it can be solved in time $O(f(k) \cdot |x|^c)$, where f is an arbitrary function and $|x|$ is the size of the input. When a problem is W-hard it might still have a polynomial time algorithm for each fixed k, for example $O(|x|^k)$, but it is very unlikely to be FPT. For a complete reference see [8].

A fundamental question, and an interesting problem on its own, is whether it is possible to recognize parametrized graph classes in FPT time. Cai [4] showed that for all classes of graphs characterized by a finite forbidden set of induced subgraphs, the corresponding parametrized classes ($+kv$, $+ke$, $-ke$) can be recognized in FPT time. In addition it has been shown that also chordal$+kv$ [22], chordal$-ke$ [20,21,4], strongly chordal$-ke$ [20], interval$-ke$ [17], proper interval$-ke$ [20,21] and planar$+kv$ graphs [24] are recognizable in FPT time. Split graphs are characterized by a finite forbidden set of induced subgraphs, hence, by the result of [4], split$+ke$, split$-ke$ and split$+kv$ graphs are recognizable in FPT time. By the same result we can assume that, given our input, we know which set of k edges or vertices we have to remove/add to get a split graph. Such a set is called a *modulator* of the graph. Related to the problem of finding a modulator efficiently, in [5] Cai asks whether there is a recognition algorithm that is not only uniformly polynomial, but uniformly linear for parametrized split graphs. We can answer affirmatively to this question.

At the moment it seems like mostly coloring problems have been investigated on parametrized graph classes. In [5], Cai shows that finding the chromatic number of split$+ke$ and split$-ke$ graphs is FPT, while it is W[1]-hard for split$+kv$ graphs. He also shows that the same problem is solvable in linear time on bipartite$+1v$ and bipartite$+2e$ graphs, but NP-complete for bipartite$+2v$ and bipartite$+3e$ graphs. Takenaga and Higashide [29] study the chromatic number problem for comparability$+ke$ graphs, and prove that it becomes NP-complete for $k \geq 2$. Finally in [23], Marx gives an FPT algorithm for coloring chordal$+ke$ graphs.

In this paper we consider various other problems that are known to be polynomially solvable on split graphs, and we study their complexity for split$+ke$ and split$+kv$ graphs. For example we study problems like minimum split completions and minimum fill-in. In these cases we want to find the minimum number of edges to be added to a graph to make it split or chordal. Minimum fill-in in particular is an extremely well studied problem with a number of practical applications (see for example [13]). Of course these problems are easy on split graphs since split graphs are chordal. That is why it comes as a great surprise that,

even though minimum split completion of split+kv graphs is solvable in FPT time, minimum fill-in becomes NP-complete even for split+$1v$ graphs. This implies that minimum fill-in of chordal+$1v$ graphs is also NP-complete. Motivated by this result we investigate also the treewidth of split+kv graphs. Threewidth is another problem related to chordal completions with important algorithmic applications (see [3] for a survey). In contrast with the previous result, we prove that treewidth of split+$1v$ graphs can be computed in polynomial time, giving the first graph class (to our knowledge) on which minimum fill-in and treewidth have different computational complexity.

It is worth mentioning that there has been some work on similar problems, but restricted to special cases: minimum split completions of split+$1v$ and $+1e$ graphs [16] and minimum cograph completions of cograph+$1e$ graphs [25].

In section 3 we give a simple FPT algorithm to list all maximal independent sets and cliques of a split+kv graphs. This leads to the FPT algorithm for computing the minimum split completion of split+kv graphs, generalizing the result of [16]. Let us also point out that all FPT algorithms given for split+kv graphs, work also for split+ke graphs, but not vice versa.

In section 4 we focus on minimum fill-in and treewidth for split+ke and split+kv graphs. As far as split+ke graphs are concerned, we are able to give an FPT algorithm that given a graph in this class, can list all its minimal triangulations. This implies that we can compute both treewidth and minimum fill-in within the same time bound. For split+kv graphs we prove the results mentioned above. As last remark, notice that in this paper we parametrize the minimum fill-in problem by the distance k of the graph class, not by the number of edges to be added as in [20,21,4]. In the latter case, in fact, the problem is FPT for general graphs.

We conclude with listing some open problems and possible directions of research. Due to limited space, the proofs of some of our results will be omitted. These results are marked with an asterisk.

2 Notation and Definitions

All graphs in this paper are simple and undirected. For a graph $G = (V, E)$, we let $n = |V|$ and $m = |E|$. The set of neighbors of a vertex $v \in V$ is denoted by $N(v)$, and the degree of a vertex v is denoted by $d(v) = |N(v)|$. The neighborhood of a set of vertices S is defined as $N(S) = \cup_{x \in S} N(x) \setminus S$. Also, $N[v] = N(v) \cup \{v\}$ and $N[S] = \cup_{x \in S} N[x]$. We distinguish between subgraphs and induced subgraphs. In this paper, a *subgraph* of $G = (V, E)$ is a graph $G_1 = (V, E_1)$ with $E_1 \subseteq E$, and a *supergraph* of G is a graph $G_2 = (V, E_2)$ with $E \subseteq E_2$. We will denote these relations informally by the notation $G_1 \subseteq G \subseteq G_2$ (proper subgraph relation is denoted by $G_1 \subset G$). An induced subgraph of $G = (V, E)$ over a set of vertices $U \subseteq V$, is the graph $G[U] = (U, E_U)$, where $E_U = \{xv \in E | x, v \in U\}$. The complement of G is denoted by $\overline{G}$. Given a vertex x of G, $G - x = G[V \setminus \{x\}]$.

A subset K of V is a *clique* if K induces a complete subgraph of G. A subset I of V is an *independent set* if no two vertices of I are adjacent in G. We use

$\omega(G)$ to denote the size of the largest clique in G, and $\alpha(G)$ to denote the size of a largest independent set in G. If there is no ambiguity, we will use only ω and α. We call a vertex v *simplicial* if $N(v)$ induces a clique. A vertex cover is a set $V' \subseteq V$ such that all edges of G are incident to at least a vertex in V'. A dominating set, is a set $V' \subseteq V$ such that $N[V'] = V$.

G is a *split graph* if there is a partition $V = I + K$ of its vertex set into an independent set I and a clique K. Such a partition is called a *split partition* of G. There is no restriction on the edges between vertices of I and vertices of K and the partition itself is not necessarily unique.

For the following result, note that a simple cycle on k vertices is denoted by C_k and that a complete graph on k vertices is denoted by K_k. Thus $2K_2$ is the graph that consists of 2 isolated edges. Also, a chordal graph is a graph that does not contain an induced subgraph isomorphic to C_k for $k \geq 4$.

Theorem 1. (Földes and Hammer [9]) *Let G be an undirected graph. The following conditions are equivalent:*

(i) G is a split graph.
(ii) G and $\overline{G}$ are chordal graphs.
(iii) G contains no induced subgraph isomorphic to $2K_2, C_4$ or C_5.

Remark 1. Every induced subgraph of a split graph is also a split graph.

A graph $G = (V, E)$ is called: A split$+ke$ graph if there is a set E_k with $E_k \subset E$ and $|E_k| \leq k$ such that $G' = (V, E \setminus E_k)$ is a split graph; A split$-ke$ graph if there is a set E_k with $E_k \cap E = \emptyset$ and $|E_k| \leq k$ such that $G' = (V, E \cup E_k)$ is a split graph; and split$+kv$ graph if there is a set $V_k \subset V$ with $|V_k| \leq k$ such that $G[V \setminus V_k]$ is a split graph. The set E_k or V_k is referred to as a *modulator* of the graph.

For a given arbitrary graph $G = (V, E)$, a split graph $H = (V, E \cup F)$, with $E \cap F = \emptyset$, is called a *split completion* of G. The edges in F are called *fill edges*. H is a *minimum* split completion of G if $|F|$ is as small as possible, while H is a *minimal* split completion of G if $(V, E \cup F')$ fails to be a split graph for every proper subset F' of F.

Minimal and minimum chordal completions are defined analogously to minimal and minimum split completions and they are also called *triangulations*. In particular the problem of making a graph chordal adding the minimum number possible of fill edges is referred to as the minimum fill-in problem. Both minimum fill-in and the minimum split completion problem are NP-hard [30,1]. For a split or chordal graph G, $\alpha(G)$ and $\omega(G)$ can be computed in linear time [10], whereas these are NP-hard problems for general graphs.

Treewidth is a parameter that measures how tree-like a graph is, and computing it is NP-hard for general graphs [28]. The formal definition involves the concept of tree decomposition.

Definition 1. *A tree decomposition of a graph $G = (V, E)$ is a pair $(\{X_i \mid i \in I\}, T = (I, M))$ where $\{X_i \mid i \in I\}$ is a collection of subsets of V (also called bags), and T is a tree such that:*

- $\bigcup_{i \in I} X_i = V$
- $(u, v) \in E \Longrightarrow \exists i \in I$ *with* $u, v \in X_i$
- *For all vertices* $v \in V$, $\{i \in I \mid v \in X_i\}$ *induces a connected subtree of* T.

The *width* of a decomposition $(\{X_i \mid i \in I\}, T = (I, M))$ is $\max_{i \in I} |X_i| - 1$. The *treewidth* of a graph G, $tw(G)$, is the minimum width over all tree decompositions of G. The treewidth of a graph can also be defined from the point of view of minimal triangulations. In particular, the treewidth of a graph G is $\min_H \omega(H) - 1$, where H is a minimal triangulation of G. So finding the treewidth of G is equivalent to finding a minimal triangulation H of G with the smallest maximum clique. Chordal graphs have a tree decomposition with minimum width called *clique tree*, where each bag corresponds to a maximal clique and each leaf-bag contains at least a simplicial vertex [2]. Also, for a chordal graph G, we have $tw(G) = \omega(G) - 1$. Finally, as mentioned in the introduction, if a problem is FPT for split+kv graphs, then it is FPT also for split+ke graphs. In fact every split+ke graph is also a split+kv graph. To see this: we can always select at most k vertices that cover all the k edges. Removing these vertices clearly gives a split graph by Remark 1. Furthermore, the complement of a split+ke graph, is a split$-ke$ graph since split graphs are self complementary, which follows from Theorem 1. Hence, if we can solve a problem for split+ke graph, we can solve its complement on split$-ke$ graph. For example if we can solve the minimum split completion problem for split+ke graphs, then we can solve the minimum split deletion problem for split$-ke$ graphs.

3 Listing Maximal Independent Sets in FPT Time

It is already known that for every hereditary family $\mathcal{C}$ for which maximum clique, maximum independent set and minimum vertex cover are polynomial time solvable, then the same problem can be solved in FPT time on the corresponding parametrized classes [5]. However in this case we are interested in listing all maximal independent sets and cliques in FPT time, not only finding the maximum. Notice that if it is possible to list all maximal independent sets or cliques in FPT time, then it is possible possible to find the corresponding maximum as well, but not vice versa. In particular it is easy to show that for every hereditary family $\mathcal{C}$ with a polynomial number of maximal independent sets (or cliques), it is possible to list in FPT time all the maximal independent sets (or cliques) of $\mathcal{C}+kv$. In this section we will prove formally such result, and apply it to split+kv graphs. This will lead to a simple FPT algorithm to solve the minimum split completion problem for this graph class.

Before to start, we would like to settle also a question left open by Cai in [5] about the existence of a uniformly linear algorithm to find a modulator for parametrized split graphs. The answer is affirmative, and follows from [16,14] where two linear time certification algorithms for split graphs are given. These algorithms, in fact, can not only recognize if a graph is split, but also return a forbidden subgraph if it is not. Once we can find a forbidden subgraph in

linear time, we can try and remove each of its vertices, and run the recognition algorithm again for at most k times in each case. Since the largest forbidden subgraph of a split graph has 5 vertices, this procedure would yield a search tree with at most $O(5^k)$ nodes and therefore a total running time of order $O(5^k \cdot (n+m))$. The same idea holds for any graph class $\mathcal{C}$ with a finite set of forbidden induced subgraphs. A modulator of the corresponding parametrized family can be found in linear time for each fixed k if, given a graph not in $\mathcal{C}$, a forbidden subgraph can be found in linear time.

The following is implicit from the results of [19].

Lemma 1. *[19] If a graph has a polynomial number of maximal independent sets, then they can be listed in polynomial time.*

Lemma 2. ** Let $\mathcal{C}$ be a hereditary family of graphs with a polynomial number of maximal independent sets. Then, given a graph $G = (V, E)$ in $\mathcal{C} + kv$ and a modulator V_k, it is possible to list all its maximal independent sets in FPT time.*

At this point it is enough to show that split graphs have a polynomial number of maximal independent sets to get the result we need.

Lemma 3. ** In a split graph there are at most $\omega + 1$ maximal independent sets and $\alpha + 1$ maximal cliques.*

Theorem 2. ** Let $G = (V, E)$ be a split+kv graph, and let V_k be a modulator of G. Then the maximal independent sets of G can be listed in time $O(2^k \cdot k^2 \cdot nm)$.*

Since all previous results on maximal independent sets hold also for maximal cliques (just consider the complement of the graph instead), we can give the following theorem.

Theorem 3. *Let $G = (V, E)$ be a split+kv graph. Then a maximum independent set, maximum clique, minimum vertex cover or minimum independent dominating set of G can be found in time $O(2^k \cdot k^2 \cdot nm)$.*

In the rest of the section we give an FPT algorithm to compute a minimum split completion of split+kv graphs, based on the following lemma.

Lemma 4. *Let $H = (V, E \cup F)$ be a minimal split completion of a graph $G = (V, E)$. Then there exists a maximal independent set $I \subseteq V$ in G such that H can be obtained making $G[V \setminus I]$ into a clique.*

Proof. Let $V = K + I$ be a split partition of H. We will prove that there exists a maximal independent set $I' \supseteq I$ of G such that all fill edges of H must be in $K \setminus I'$, hence proving the statement. First of all notice that in H no fill edges can be incident to vertices of I, otherwise they could be removed to produce a split completion $H' = (V, E \cup F')$ of G with $F' \subset F$, contradicting that H is minimal. Now assume for the sake of contradiction that for every possible maximal independent set I' of G, there are some fill edges incident to vertices of I' in H. If we remove all such fill edges, we get a graph $H' = (V, E \cup F')$

with a valid split partition $V = K \setminus I' + I'$. However this is a contradiction, because it implies that there exists a subgraph of H that is a split completion of G, contradicting the minimality of H. We can conclude that there must exist a maximal independent set I' of G, such that H can be obtained adding fill edges only in $G[V \setminus I']$.

Notice that it is not true, however, that a minimum split completion of a graph $G = (V, E)$ can be obtained taking some maximum independent set I of G and making $G[V \setminus I]$ into a clique. Actually we cannot even guarantee that taking a maximum independent set we can produce a minimal split completion (consider a path on 4 vertices). That is why we need to list all maximal independent sets of a graph to find a minimum split completion of it.

Theorem 4. * *Let $G = (V, E)$ be a split+kv graph. Then a minimum split completion of G can be computed in time $O(2^k \cdot k^2 \cdot nm)$.*

4 Minimum Fill-In and Treewidth

In this section we give an FPT algorithm for minimum fill-in and treewith of split+ke graphs, and show that the same problem is harder for split+kv graphs. In particular we prove that there exists a graph class for which minimum fill-in is NP-complete while treewidth is polynomial, namely split+$1v$ graphs.

4.1 Split+ke Graphs

In this section we will use the connection between the Elimination Game and chordal graphs [27]. Running the Elimination Game on a graph $G = (V, E)$ and an ordering β of its vertices, means to remove the vertices from G in the order given by β so that, after removing a vertex, we make its neighborhood in the current graph into a clique. It is well known that this produces a triangulation of G. In particular for each minimal triangulation H of a graph G, there exists an ordering β that can produce it [26]. An ordering is called *perfect elimination ordering* if every vertex is simplicial when it is deleted during the Elimination Game. Chordal graphs are exactly the graphs that admit a perfect elimination ordering.

We will show that for split+ke graphs, we can produce in FPT time all elimination orderings that produce a minimal triangulation.

The results in Observation 1 and 2 follow from previous results on chordal graphs and minimal triangulations, for further references see [13].

Observation 1. *Given a graph $G = (V, E)$, let $K \subseteq V$ be a set of vertices such that $G[K]$ is a clique. Then every minimal triangulation H of G can be obtained by running the Elimination Game on G and an elimination ordering where the vertices of K are eliminated at the end, in any order.*

Observation 2. *Let H be a minimal chordal completion of an arbitrary graph G. No simplicial vertex of G is incident to any fill edge in H.*

Using the previous two Lemmas, we will show that all minimal chordal completion of a split+ke graph, can be obtained considering all the permutations of at most $2k$ vertices.

Theorem 5. *Let $G = (V, E)$ be a split+ke graph, and let the set $E_k \subset E$ be a modulator for G. Then all minimal chordal completions of G can be computed in $O((2k)! \cdot nm)$ time.*

Proof. Let I be a maximum independent set of the split graph $G' = (V, E \setminus E_k)$. Notice that all vertices in I are simplicial in G'. Let K be the clique $G'[V \setminus I]$. In G, all vertices of I that are not incident to an edge of E_k are still simplicial. Hence, since $|E_k| \leq k$, there can be at most $2k$ non simplicial vertices in G that belong to I. Let us call this set S. By Observation 2, we can remove all vertices in $I \setminus S$ and consider only $G_S = G[S \cup K]$. $G_S[K]$ is a clique, so by Observation 1 all elimination orderings of G_S can be produced by considering only the orderings of the vertices in S. This means that there are at most $(2k)!$ meaningful orderings, and for each of them it takes $O(nm)$ time to both compute the corresponding triangulation and check whether it is minimal [6].

Corollary 1. *Minimum fill-in and treewidth of split+ke graphs can be computed in time $O((2k)! \cdot nm)$.*

4.2 Split+kv Graphs

Minimum Fill-in. Here we show that adding the minimum number of edges to make a graph that is split+kv into chordal, is NP-complete even for $k = 1$. In order to prove this result, we give a reduction from the minimum fill-in for co-bipartite graphs, that was shown NP-complete by Yannakakis in [30].

Let $G = (P, Q, E)$ be a co-bipartite graph, where P and Q are cliques. From G we build a new graph $G_S = (P \cup Q \cup C \cup \{x\}, E_S)$ in the following way. Take a copy of G. Remove all edges between vertices in P. Create a clique C of size $|P|(|P| - 1)/2 + |P| \cdot |Q| + 1$ and add all edges between the vertices in C and $P \cup Q$. Finally add a vertex x and make it universal to P. Since we can partition G_S into a clique $C \cup Q$, an independent set P and a vertex x, we have the following observation.

Observation 3. *The graph G_S is split+$1v$.*

Lemma 5. *In every minimum triangulation H_S of $G_S = (P \cup Q \cup C \cup \{x\}, E_S)$, $H_S[P]$ is a clique.*

Proof. First of all notice that there is a trivial upper bound for the size of a minimum triangulation of G_S, namely $|P|(|P| - 1)/2 + |P| \cdot |Q|$. That is, make $G_S[P \cup Q]$ into a clique. Furthermore, for every pair of vertices $p_1, p_2 \in P$ and every vertex $c \in C$, $G_S[\{x, p_1, c, p_2\}]$ is a C_4, and there are only two ways to kill such a cycle: either add the edge p_1p_2, or make x universal for C. We will prove the statement by contradiction. Assume there exists a minimum triangulation

H'_S of G_S where $H'_S[P]$ is not a clique. Then x must be universal to C or, by the previous discussion, for any two non adjacent vertices in P and a vertex of C not incident to x, we would have a C_4 in H'_S. Notice that making x universal to C requires the addition of $|C|$ fill edges. However, by the construction of G_S, $|C| = |P|(|P|-1)/2 + |P| \cdot |Q| + 1 > |P|(|P|-1)/2 + |P| \cdot |Q|$. This shows that H'_S cannot be a minimum triangulation of G_S, since it exceeds the upper bound we gave previously.

Lemma 6. *A minimum triangulation of a co-bipartite graph $G = (P, Q, E)$ can be obtained adding a set F of fill edges to G if and only if a minimum triangulation of $G_S = (P \cup Q \cup C \cup \{x\}, E_S)$ can be obtained adding the set of fill edges F to G_S and making $G_S[P]$ into a clique.*

Proof. From Lemma 5 we know that in every minimum triangulation of G_S, the subgraph induced by P is a clique. Let us then consider the graph G'_S, namely the graph G_S where $G_S[P]$ has been made into a clique. In G'_S the vertex x is simplicial, hence by Observation 2 there will be no fill edge incident to it. This also implies that no fill edges can be incident to any vertex of C, because they are universal for everything but x. We can conclude that finding a minimum triangulation of G_S is equivalent to finding a triangulation of $G'_S[P \cup Q]$. The fact that $G'_S[P \cup Q] = G$ concludes the proof.

Given the previous Lemma and the fact that G_S can be built in polynomial time, we can state the main theorem of this section.

Theorem 6. *The minimum fill-in problem for split+kv is NP-complete for $k \geq 1$.*

Corollary 2. *The minimum fill-in problem for chordal+kv is NP-complete for $k \geq 1$.*

Treewidth. In contrast to the minimum fill-in problem, we show that the treewidth of split+kv can be found in polynomial time when $k = 1$. We have strong evidences that this still holds for $k = 2$, but for larger values of k we only conjecture the existence of a polynomial algorithm, probably not FPT.

Let $G = (V, E)$ be a split+kv graph with modulator V_k and let us define $\omega = \omega(G[V \setminus V_k])$. Since split graphs are chordal, we know that $tw(G[V \setminus V_k]) = \omega - 1$. Besides, adding k vertices to a graph cannot increase the treewidth by more than k, therefore we have that $\omega - 1 \leq tw(G) \leq \omega + k - 1$. Let us now consider $G = (V, E)$ to be a split+$1v$ graph, $V_k = \{x\}$ its modulator and again $\omega = \omega(G - x)$. Then we have the following observation.

Observation 4. *The treewidth of split+$1v$ graph is either $\omega - 1$ or ω.*

For the rest of the section we assume G not to be chordal and $\omega = \omega(G - x) = \omega(G)$. Also, we define G' to be the graph obtained from G removing recursively all simplicial vertices not in $N_G[x]$. We will prove that computing the threewidth of G is equivalent to computing the size of the maximum clique of $G' - x$.

Observation 5. *All simplicial vertices removed to obtain G' have degree at most $\omega - 1$.*

Lemma 7. *$tw(G') < \omega$ if and only if $tw(G) < \omega$.*

Proof. If $tw(G) < \omega$, it is straightforward to see that $tw(G') < \omega$, since G' is an induced subgraph of G. For the other direction, let H' be a minimal triangulation of G' with treewidth $< \omega$. Let us add back to H' the simplicial vertices removed to obtained G' and call the resulting graph H. Notice that H is still chordal, namely a chordal completion of G, and it does not contain cliques of size greater than ω. In fact $\omega(H') \leq \omega$ and the vertices we put back in order to get H do not have degree greater than $\omega - 1$ by Observation 5, so they cannot create a clique of size greater than ω. This means that there exists a chordal completion of G with maximum clique size not greater than ω. Hence $tw(G) < \omega$.

Lemma 8. *Let $\omega' = \omega(G' - x)$. Then $tw(G') = \omega'$.*

Proof. If $\omega(G') = \omega' + 1$ the result follows directly by Observation 4 since G' is still a split+$1v$ graph, otherwise take a split partition $K + I$ of $G' - x$, such that K is a clique of size ω'. Notice that x is adjacent to all I, or there would be simplicial vertices not adjacent to x. Let us assume for the sake of contradiction that the treewidth of G' is not ω'. Then it must be $\omega' - 1$ by Observation 4. Take a minimal triangulation H of G' with treewidth $\omega' - 1$, and a clique tree T_H of H. In H there cannot be cliques of size greater than ω'. This means that there must be a bag of T_H containing the whole clique K, and nothing else, since every clique must be completely contained in some bag of the tree decomposition. Assume such bag is an internal bag. Since H cannot be a complete graph, every internal bag of its tree decomposition is a separator in the graph [18]. However K cannot separate any two vertices in the graph, since $H[V \setminus K] = H[x \cup I]$ is a connected graph, as $G'[x \cup I]$ was. Hence the bag containing K must be a leaf of T_H. This implies that there exists a simplicial vertex $v \in H[K]$. Since G' has no simplicial vertices that are not neighbors of x, every vertex in K must have at least a neighbor in $I \cup \{x\}$. Hence $N_H[v] \supset H[K]$, meaning that there is a clique of size greater than ω' in H, giving a contradiction.

We are now ready to give the main Theorem.

Theorem 7. *Treewidth is polynomial for split+$1v$ graphs.*

Proof. Given a split+$1v$ graph G with modulator $\{x\}$, we can check whether it is chordal in linear time [13] and, if so, output $tw(G) = \omega(G) - 1$. Otherwise, if $\omega(G - x) + 1 = \omega(G)$ we output $tw(G) = \omega(G - x)$ by Observation 4. Notice that we can find $\omega(G)$ in polynomial time by Lemma 3. If none of the previous holds, we can apply Lemma 7 and 8. That is, we need to find $\omega(G' - x)$ and output $tw(G) = \omega(G) - 1$ if $\omega(G' - x) < \omega(G)$, or $tw(G) = \omega(G)$ otherwise. Since G' and $\omega(G' - x)$ can be found in polynomial time, the result follows.

5 Conclusions

In this paper we studied how some problems that are easy on split graphs behave when the input graph is slightly modify by the addition of some edges or vertices. The results were in part counter-intuitive and surprising, and this gives strong motivation to continue studying these parametrized graph classes. The next natural step would be to take in consideration parametrized chordal graphs, as they are a very important superclass of split graphs. For example it would be very interesting to know whether there is an FPT algorithm for minimum fill-in and treewidth for chordal+ke like for split graphs. For chordal+kv, instead, we proved that minimum fill-in is NP-complete, but we do not know much about treewidth, even though it seems polynomial for $k = 1$. However it would be maybe easier to try and settle the problem for split+kv when $k \geq 2$, since any hardness result would still hold for chordal+kv.

Acknowledgments. I would like to thank Yngve Villanger and Daniel Lokshtanov for the discussion of the problem and the many useful suggestions, and my supervisor Pinar Heggernes for helping with the revision of the paper.

References

1. Shamir, R., Natanzon, A., Sharan, R.: Complexity classification of some edge modification problems. Discrete Applied Mathematics 113(1), 109–128 (2001)
2. Blair, J.R.S., Peyton, B.: An introduction to chordal graphs and clique trees. In: Graph Theory and Sparse Matrix Computations, pp. 1–29. Springer, Heidelberg (1993)
3. Bodlaender, H.L.: Discovering treewidth. In: Vojtáš, P., Bieliková, M., Charron-Bost, B., Sýkora, O. (eds.) SOFSEM 2005. LNCS, vol. 3381, pp. 1–16. Springer, Heidelberg (2005)
4. Cai, L.: Fixed-parameter tractability of graph modification problems for hereditary properties. Inf. Process. Lett. 58(4), 171–176 (1996)
5. Cai, L.: Parameterized complexity of vertex colouring. Discrete Applied Mathematics 127(3), 415–429 (2003)
6. Dahlhaus, E.: Minimal elimination ordering inside a given chordal graph. In: Möhring, R.H. (ed.) WG 1997. LNCS, vol. 1335, pp. 132–143. Springer, Heidelberg (1997)
7. Dirac, G.A.: On rigid circuit graphs. In: Abhandlungen aus dem Mathematischen Seminar der Universiat Hamburg, vol. 25, pp. 71–75 (1961)
8. Downey, R.G., Fellows, M.R.: Parameterized Complexity. Springer, Heidelberg (1999)
9. Foldes, S., Hammer, P.L.: Split graphs. In: 8th South-Eastern Conference on Combinatorics, Graph Theory and Computing, pp. 311–315 (1977)
10. Golumbic, M.C.: Algorithmic Graph Theory and Perfect Graphs, 1st edn. Academic Press, London (1980)
11. Guo, J., Hüffner, F., Niedermeier, R.: A structural view on parameterizing problems: Distance from triviality. In: Downey, R.G., Fellows, M.R., Dehne, F. (eds.) IWPEC 2004. LNCS, vol. 3162, pp. 162–173. Springer, Heidelberg (2004)

12. Hammer, P.L., Simeone, B.: The splittance of a graph. Combinatorica 1(3), 275–284 (1981)
13. Heggernes, P.: Minimal triangulations of graphs: A survey. Discrete Mathematics 306(3), 297–317 (2006)
14. Heggernes, P., Kratsch, D.: Linear-time certifying algorithms for recognizing split graphs and related graph classes. Technical report, 2006-328, UiB (2006)
15. Heggernes, P., Mancini, F.: Minimal split completions of graphs. In: Correa, J.R., Hevia, A., Kiwi, M. (eds.) LATIN 2006. LNCS, vol. 3887, pp. 592–604. Springer, Heidelberg (2006)
16. Heggernes, P., Mancini, F.: A completely dynamic algorithm for split graphs. Technical report, 2006-334, UiB (2006)
17. Heggernes, P., Paul, C., Telle, J.A., Villanger, Y.: Interval completion is fixed parameter tractable. In: Proceedings of the 39th Annual ACM Symposium on Theory of Computing, STOC, pp. 374–381 (2007)
18. Ho, C.W., Lee, R.C.T.: Counting clique trees and computing perfect elimination schemes in parallel. Inf. Process. Lett. 31(2), 61–68 (1989)
19. Johnson, D.S., Papadimitriou, C.H.: On generating all maximal independent sets. Inf. Process. Lett. 27(3), 119–123 (1988)
20. Kaplan, H., Shamir, R., Tarjan, R.E.: Tractability of parameterized completion problems on chordal, strongly chordal, and proper interval graphs. SIAM J. Comput. 28(5), 1906–1922 (1999)
21. Kaplan, H., Shamir, R., Endre, R.: Tractability of parameterized completion problems on chordal and interval graphs: Minimum fill-in and physical mapping. In: 35th Annual Symposium on Foundations of Computer Science, FOCS 2004, pp. 780–791. IEEE, Los Alamitos (2004)
22. Marx, D.: Chordal deletion is fixed-parameter tractable. In: Fomin, F.V. (ed.) WG 2006. LNCS, vol. 4271, pp. 37–48. Springer, Heidelberg (2006)
23. Marx, D.: Parameterized coloring problems on chordal graphs. Theor. Comput. Sci. 351(3), 407–424 (2006)
24. Marx, D., Schlotter, I.: Obtaining a planar graph by vertex deletion. In: WG, LNCS (to appear, 2007)
25. Nikolopoulos, S.D., Palios, L.: Adding an edge in a cograph. In: Kratsch, D. (ed.) WG 2005. LNCS, vol. 3787, pp. 214–226. Springer, Heidelberg (2005)
26. Ohtsuki, T., Cheung, L.K., Fujisawa, T.: Minimal triangulation of a graph and optimal pivoting ordering in a sparse matrix. J. Math. Anal. Appl. 54, 622–633 (1976)
27. Parter, S.: The use of linear graphs in gauss elimination. SIAM Review 3(2), 119–130 (1961)
28. Corneil, D.G., Arnborg, S., Proskurowski, A.: Complexity of finding embeddings in a k-tree. SIAM J. Algebraic and Discrete Methods 8, 277–284 (1987)
29. Takenaga, Y., Higashide, K.: Vertex coloring of comparability+ke and -ke graphs. In: Fomin, F.V. (ed.) WG 2006. LNCS, vol. 4271, pp. 102–112. Springer, Heidelberg (2006)
30. Yannakakis, M.: Computing the minimum fill-in is np-complete. SIAM Journal on Algebraic and Discrete Methods 2(1), 77–79 (1981)

Weighted Treewidth Algorithmic Techniques and Results*

Emgad Bachoore and Hans L. Bodlaender

Department of Information and Computing Sciences,
Utrecht University, P.O. Box 80.089, 3508 TB Utrecht, The Netherlands
bachoore@cs.uu.nl, hansb@cs.uu.nl

Abstract. From the analysis of algorithms for probabilistic networks, it is known that a tree decomposition of the minimum treewidth may not be optimal for these algorithms. Instead of treewidth, we consider therefore the weighted treewidth of a weighted graph. In this paper, we present a number of heuristics for determining upper and lower bounds on the weighted treewidth, and a branch and bound algorithm for finding the exact weighted treewidth for weighted graphs.

1 Introduction

In many graphical models and networks, each vertex is associated with a weight. The weights of the vertices in the graph or network play a significant role for finding an optimal solution for the problem. Triangulation of Bayesian Networks of probabilistic networks and logical partitioning approaches are examples of such problems.

Many decision support systems have probabilistic networks as underlying technology [11]. In these networks, we model dependencies and independencies between statistical variables using a directed acyclic graph. Each statistical variable is represented by a a vertex in the network. An important problem on these networks is probabilistic inference: we want to find the probability distribution for a variable, given a value assignment to some other variables. An efficient algorithm for inference is based on a tree decomposition of the moralized graph of the network, since this graph appears to have small treewidth for many probabilistic networks that model real-life situation. See for details [8,9,10].

In order to find a tree decomposition on which the algorithm from [9,10] for inference takes little time, we search of a tree decomposition of small *weighted width*. The time this algorithm needs to process one bag of the tree decomposition is proportional to the product over the variables, represented by the vertices in the bag, of the number of different values that the variable can assume. If all values in the probabilistic network can assume the same number c of different values (e.g., all are binary and $c = 2$), then the time of the Lauritzen-Spiegelhalter algorithm is bounded by $O(c^k \cdot n)$. However, in practice, the statistical variables in a probabilistic networks may have different numbers of possible values, and thus the tree decomposition of minimum width may not be optimal for this algorithm. Thus, we look for a tree decomposition with minimum weighted width instead of one of minimum width.

* This work has been supported by the Netherlands Organization for Scientific Research NWO (project TACO: 'Treewidth And Combinatorial Optimization').

T. Tokuyama (Ed.): ISAAC 2007, LNCS 4835, pp. 893–903, 2007.

The problem of finding the exact weighted treewidth of a weighted graph is an NP hard problem, even if all weights are equal [1]. Therefore, we introduce, in this paper, besides a weighted variant of the branch and bound algorithm for treewidth from [3], some heuristics for finding lower and upper bounds on weighted treewidth.

Several proofs are skipped in this extended abstract due to space constraints. For more details see the techniqual report in http://www.cs.uu.nl/staff/bachoore, [4].

2 Preliminaries

We denote an undirected weighted graph by $G = (V, E, w)$, where

$w : V \rightarrow N^+$ is a weight function. We refer to the weight of a vertex v in a weighted graph G as $w_G(v)$, or in short $w(v)$. We denote the set of neighbors of vertex v by $N(v)$, and the set of neighbors of v plus v itself by $N[v]$. A vertex v in G is called ***simplicial***, if its set of neighbors $N(v)$ forms a clique in G. A vertex v in G is called ***almost simplicial***, if its neighbors except one form a clique in G, i.e., if v has a neighbor w such that $N(v) - \{w\}$ is a clique. A set of vertices W for which there is an $x \in W$ with $W - \{x\}$ a clique is called an ***almost clique***, x is called the ***excluding vertex***. A graph G is called ***triangulated*** (or: chordal) if every cycle of length four of more possesses a chord. A ***chord*** is an edge between two non consecutive vertices of the cycle. A graph $G = (V, E)$ is a subgraph of graph $H = (W, F)$ if $V \subseteq W$ and $E \subseteq F$. A graph $H = (V, F)$ is a ***triangulation of graph*** $G = (V, E)$, if G is a subgraph of H and H is a triangulated graph. A ***linear ordering*** of a graph $G = (V, E)$ is a bijection $f : V \rightarrow \{1, 2, \cdots, |V|\}$. A linear ordering of the vertices of a graph G, $\sigma = [v_1, \cdots, v_n]$ is called a ***perfect elimination order (p.e.o.)*** of G, if for every $1 \leq i \leq n$, v_i is a simplicial vertex in $G[v_{i+1}, \cdots, v_n]$, i.e., the higher numbered neighbors of v_i form a clique. It has been shown in [8] that a graph G is triangulated, if and only if G has a $p.e.o.$ The weight of a set of vertices of a graph G, $S \subseteq V$ is $w(S) = \prod_{v \in S} w(v)$. The ***total weight of a graph*** G equals the weight of the set of its vertices, $w(V) = \prod_{v \in V} w(v)$. The ***neighborhood weight*** of a vertex v in a graph G, $nw_G(v) = \prod_{v \in N[v]} w(v)$, or in short $nw(v)$.

The definition of a tree decomposition of a weighted graph $G = (V, E, w)$ is exactly the same as for unweighted graphs. A ***tree decomposition*** of $G = (V, E, w)$ is a pair $(\{X_i \mid i \in I\}, T = (I, F))$ with $\{X_i \mid i \in I\}$ a collection of subsets of V and T a tree, such that $\bigcup_{i \in I} X_i = V$, for all $\{v, w\} \in E$, there is an $i \in I$ with $v, w \in X_i$, and for each v, $\{i \in I \mid v \in X_i\}$ forms a connected subtree of T. The ***weighted width*** of a tree decomposition equals $\max_{i \in I} w(X_i)$

and the ***weighted treewidth*** of a graph G, $wtw(G)$, is the minimum weighted width over all tree decompositions of G.

A ***minor*** graph $G' = (W, F, w')$ of a weighted graph $G = (V, E, w)$ is a weighted graph obtained from G by a sequence of zero or more vertex removals, edge removals, and / or edge contractions, where an edge contraction for weighted graphs is the operation, that given an edge $\{x, y\} \in E$, removes x and y and their incident edges from G and adds a new vertex z, adjacent to the vertices that were adjacent to x or y, with the weight of z equal to $\min(w(x), w(y))$. The ***fill-in*** of a vertex x in a weighted graph G, *fill-in*$_G(x)$, or in short, *fill-in*(x), is the number of edges that

must be added between the neighbors of x to make it simplicial, i.e., *fill-in*$(x) = |\{\{v,w\}|v,w \in neighbors(x), \{v,w\} \notin E\}|$. The ***fill-in excluding one neighbor*** of a vertex x in a weighted graph G, ***fill-in-excl-one***$_G(x)$, or in short, *fill-in-excl-one*(x), is the *minimum* number of edges that must be added between the neighbors of x to make it almost simplicial, i.e., *fill-in-excl-one*$(x) = \min_{z \in neighbors(x)} |\{\{v,w\}|v,w \in neighbors(x) - \{z\}, \{v,w\} \notin E\}|$.

Lemma 1. *(See [7,8]). The weighted treewidth of a graph G is the minimum $k \geq 0$ such that G is a subgraph of a triangulated graph with all cliques of weight at most k.*

Lemma 2. *(See [8]). Let G and G' be two weighted graphs. If G' be a minor of G, then the weighted treewidth of G', $wtw(G')$, is at most the weighted treewidth of G, $wtw(G)$.*

Lemma 3. *(See e.g., [5].)*

1. *For every triangulated graph $G = (V,E)$, there exists a tree decomposition $(X = \{X_i|i \in I\}, T = (I,F))$ of G, such that every set X_i forms a clique in G, and for every maximal clique $W \subseteq V$, there exists an $i \in I$ with $W = X_i$.*
2. *Let $(X = \{X_i|i \in I\}, T = (I,F))$ be a tree decomposition of G of width at most k. The graph $H = (V, E \cup E')$, with $E' = \{\{v,w\}|\, \exists i \in I : v,w \in X_i\}$, obtained by making every set X_i a clique, is triangulated, and has maximum clique size at most $k+1$.*
3. *Let $(X = \{X_i|i \in I\},\ T = (I,F))$ be a tree decomposition of G, and let $W \subseteq V$ form a clique in G. Then there exist an $i \in I$ with $W \subseteq X_i$.*

3 Algorithms for Finding Weighted Treewidth

In this section, we first introduce two heuristics for obtaining a lower bound on the weighted treewidth. Then we introduce a number of heuristics for obtaining an upper bound on weighted treewidth. Finally, we present the weighted variant of the branch and bound algorithm for treewidth from [3].

3.1 Lower Bound Heuristics for Weighted Treewidth

In the following two lemmas, we give two simple lower bounds on the weighted treewidth of a graph. Later, we generalize two known heuristics for a lower bound on the treewidth, namely, Maximum Minimum Degree and the weighted variant of the Ramachandramurthi graph parameter.

Lemma 4. *Let G be a weighted graph, $wtw(G) \geq \max_{v \in V}(w(v))$.*

Lemma 5. *(Eijkhof et al. [8]). Let G be a weighted graph. $wtw(G) \geq \min_{v \in V}(nw(v))$.*

The Maximum Minimum Neighborhood Weight Heuristic, Variant Lower Bound (MMNW_lb). This heuristic builds upon Lemma 5. Given a weighted graph $G = (V, E, w)$. The algorithm works as follows: In the first step, the lower bound on the weighted treewidth is initialized with zero, $lb = 0$. Then we repeat the following operations until the graph becomes empty: Let $v \in V$ be the vertex of the minimum neighborhood weight in G; set the lower bound on the weighted treewidth to the maximum of its current value and the minimum neighborhood weight in the graph, $lb \leftarrow max(lb, nw(v))$, and remove v and its incident edges from G.

Lemma 6. *The weighted treewidth of a weighted graph G is at least the lower bound obtained from the MMNW_lb heuristic, applied to G.*

The Weighted $\gamma_w(G)$ Parameter Heuristic. Ramachandramurthi [12] introduced the graph parameter $\gamma(G)$. Let G be an unweighted graph, $\gamma(G) = \min(n-1,$ $\min_{v,w \in V, \{v,w\} \notin E}(\max(degree(v), degree(w)))$, i.e., $\gamma(G) = n - 1$, if G is a clique.

Lemma 7. *(See Ramachandramurthi [12]).*
For every graph G, $tw(G) \geq \gamma(G)$.

The weighted variant of γ, γ_w is defined as follows.

Definition 1. *For each weighted graph G,*

$$\gamma_w(G) = \min(\prod_{v \in V}(w(v)), \min_{v,w \in V, \{v,w\} \notin E} \max(nw(v), nw(w)))$$

Note that $\gamma_w(G) = \prod_{v \in V}(w(v))$, if G is a clique.

Lemma 8. *For every weighted graph G, $wtw(G) \geq \gamma_w(G)$.*

The Weighted $\gamma_w(G)$ Heuristic consists of two main steps. In the first step, we also initialize the lower bound on the weighted treewidth with zero, i.e., $lb = 0$. In the second step, if the graph is a clique, then we set lb to the maximum of its current value and the weight of the graph. Otherwise, we compute the minimum over all edges of the maximum weight of an endpoint of the edge. We set lb to the maximum of its current value and this minimum, and repeat on the graph G', obtained by removing the vertex that yielded this minimum value and its incident edges. As we compute at each step $\gamma_w(G')$ for a subgraph of G, we obtain a lower bound on the weighted treewidth of G.

3.2 Upper Bound Heuristics for Weighted Treewidth

The following lemma gives a very primitive upper bound on the weighted treewidth.

Lemma 9. *The weighted treewidth of a weighted graph G is at most $\prod_{v \in V} w(v)$.*

In the following, we present a number of heuristics for the upper bound on the weighted treewidth. All these heuristics depend basically on building a triangulation for a given graph. The following lemma is a weighted variant of a well known result for treewidth, and can be proved in the same way as the unweighted case, see e.g., [5].

Lemma 10. *Let G be a triangulated graph. The weighted treewidth of H equals the maximum weight over all maximal cliques $Q = (W, F)$ in G of $\prod_{w \in W} w(v)$.*

For building a triangulation $H_\sigma = (V, F, w)$ for a weighted graph $G = (V, E, w)$, one can use a linear ordering σ of the vertices of G such that σ is a perfect elimination ordering (p.e.o.) of H_σ, in the following way. For $i = \sigma[1], \cdots, \sigma[|V|]$, in that order, we add an edge between every pair of non-adjacent neighbors of v_i that are after v_i in the ordering, v_i is the $i'th$ vertex in σ. Thus, σ is a p.e.o. of the resulting graph H_σ. As H_σ is triangulated, its weighted treewidth equals the maximum weight of the set of vertices $S \subseteq V$, whose vertices form a maximal clique in H_σ. See Lemma 10. Hence, the weighted treewidth of H_σ, $wtw_{H_\sigma} = \prod_{v \in S} w(v)$. The following result can also be derived in the same way as the unweighted case.

Lemma 11. *There is at least one linear ordering σ for a weighted graph $G = (V, E, w)$ where we obtain the exact weighted treewidth of G.*

This suggests the general scheme in Figure 1 for the upper bound heuristics on the weighted treewidth.

set $G' \leftarrow G$; $i \leftarrow 1$; $\sigma \leftarrow ()$; $ub \leftarrow 0$;
while G' is not the empty graph
    select according to some condition a vertex v from G';
    set $ub \leftarrow \max(ub, nw(v_{G'}))$;
    eliminate v; /* remove v and turn its neighbors into a clique */
    add v to position i in the ordering σ;
    set $i \leftarrow i + 1$;
{Now ub is an upper bound on the weighted treewidth of G.}

Fig. 1. A general scheme for the upper bound heuristics on the treewidth

We call a graph G' encountered during the algorithm a temporary graph. Thus, the main differences between the following heuristics are in their conditions for selecting a vertex v from G', at each step where we have to eliminate a vertex from the graph. Therefore, we will limit our discussion over these heuristics by giving the selection conditions of each heuristic. The following two lemmas are the common factors between the selection conditions of these heuristics.

Lemma 12. *(See [8]). Let v be a simplicial vertex in a weighted graph $G = (V, E, w)$. Then the weighted treewidth of G is at least the neighborhood weight of v, $nw(v)$.*

Definition 2. *We call to a vertex v in a weighted graph G a **strongly almost simplicial** if and only if t is almost simplicial with $N(v) - \{x\}$ a clique, the neighborhood weight of v is at most the weighted treewidth of G, $nw(v) \leq wtw(G)$, and the weight of x is at most the weight of v, $w(x) \leq w(v)$.*

Lemma 13. *(See [8]). Let v be a weighted strongly almost simplicial vertex in a weighted graph $G = (V, E, w)$. Then the weighted treewidth of G is at least the neighborhood weight of v, $nw(v)$.*

In [6], the notion of ***safe reduction rule*** was introduced. A safe reduction rule rewrites a graph G to a smaller one, G', and maintains a lower bound variable low, such that the maximum of low and the treewidth of the graph at hand stays invariant, i.e., rule R is safe, if for all graphs $G,\ G'$, and all integers $low,\ low'$, we have $\max(low, wtw(G)) = \max(low', wtw(G'))$. The *Simplicial and Strongly Almost Simplicial Rules* are examples of safe rules that we have used in the selection conditions of our heuristics.

The Maximum Minimum Neighborhood Weight Heuristic, Upper Bound Variant (MMNW_ub). The main steps of this heuristic are the same as in MMNW_lb with one step more. Namely, before we remove a vertex from the graph, we add an edge between every two non adjacent neighbors of that vertex. In other words, we make a clique from the neighborhood of that vertex. Hence, the selection conditions we used in this heuristic are based upon selecting at each step when eliminate a vertex from a temporary graph, the vertex with the minimum neighborhood weight in this graph.

Lemma 14. *The weighted treewidth of a graph $G = (V, E, w)$ is at most the output of the MMNW heuristic, applied to G.*

The Minimum Fill-in Heuristic, Variant Weighted (MF_W). In this variant of the Minimum Fill-in Heuristic, we compute an upper bound on the weighted treewidth in the same manner as in the MMNW_ub heuristic, with one exception: in the selection conditions of this heuristic, we select the vertex with minimum fill-in in the temporary graph instead of the vertex with the minimum neighborhood weight as in MMNW_ub.

Lemma 15. *The weighted treewidth of a graph $G = (V, E, w)$ is at most the output of the MF_W heuristic, applied to G.*

The Minimum Fill-in Excluding One Heuristic, Variant Weighted (WMFEO). We have developed three versions of this heuristic. The differences between these are in the sequences of the conditions we use for selecting the vertex we have to eliminate from the temporary graph. In all three versions of this heuristic, and at any step when we have to eliminate a vertex from the temporary graph, we check if this graph contains any simplicial or strongly almost simplicial vertices. We eliminate these vertices from the graph, if they exist, and set the upper bound on the weighted treewidth to the maximum of its current value and the maximum neighborhood weight of these vertices. Otherwise, depending on the version, we select a vertex as follows. In the first version, WMFEO1, we perform this check: Let p be a vertex with minimum fillin in a temporary graph $G' = (W, F)$ of a given graph G. We select a vertex $q \in W$ such that, $q \neq p$, *fill-in-excl-one(q)* $\leq$ *fill-in(p)*, $nw(q) \leq low$, and $w(x) \leq w(q)$, where x is the excluded neighbor of q and low is a lower bound on the weighted treewidth of G. If more than one vertex q satisfies to these conditions, then we select the vertex of the minimum *fill-in* among them, but if still there is more than one vertex with these specifications, then we select the first vertex of the minimum *fill-in-excl-one* among these.

In version 2 of the algorithm, WMFEO2, the ties are broken using the *fill-in* and *fill-in-excl-one* in the reverse order. If more than one vertex q satisfies to the following conditions, namely, *fill-in-excl-one(q)* $<$ *fill-in(p)*, $nw(q) < low$ and $w(x) \leq w(q)$, then the vertex with minimum *fill-in-excl-one* amongst these is eliminated first. But, if there still is more than one vertex that satisfies the last condition, then the vertex with the minimum *fill-in* amongst these should be processed first.

In version 3 of the algorithm, WMFEO3, the ties are broken using the neighborhood weight besides the fill-in and fill-in-excl-one. If more than one vertex q satisfies to the two conditions, *fill-in-excl-one(q)* $<$ *fill-in(p)*, $nw(q) < low$ and $w(x) \leq w(q)$, then the vertex with minimum neighborhood weight amongst these is eliminated first.

Lemma 16. *Let ub be the upper bound on the weighted treewidth obtained from applying WMFEO1, WMFEO2, or WMFEO3 to a graph G. Then $wtw(G) \leq ub$.*

The Ratio Heuristic, Weighted Variant (WRATIO). We have adapted the two versions of the Ratio heuristic from [2] for weighted treewidth. The rules we use for selecting a vertex that we should eliminate are as follows: Again, as long as there are safe vertices in the temporary graph, namely, simplicial and/or strongly almost simplicial vertices, we eliminate these first, and set the upper bound to the maximum of its current value and the maximum neighborhood weight of these vertices. After that, each version of the heuristic proceeds in a different way. In version 1, we proceed as follows: Let p be a vertex with the minimum *fill-in* in the temporary graph G' of a graph G. A vertex $w \neq p$ in the temporary graph is selected if the *fill-in-excl-one* of w is less than or equal to the *fill-in* of p, its neighborhood weight is at most a lower bound on the weighted treewidth of G, $nw(v) \leq wtw(G)$, the $w(v) \leq w(x)$, where x is the excluded neighbor of w, and it satisfies the following condition. Let $r_1(w) =$ *fill-in-excl-one(w) / fill-in(p)*, and $r_2(w) = nw(w)/nw(p)$. We now require that $r_1(w) < r_2(w)$ to be a candidate for selection at this point. If we have more than one such candidate, we select from these a vertex with the minimum difference between r_1 and r_2, $(r_1 - r_2)$.

In version 2 of this heuristic, we proceed as follows: For all $w \in W(G')$, we select the vertex of the minimum ratio $r(w) =$ *fill-in(w)* / $nw(w)$, $(nw(w) > 1)$ amongst all vertices of G'.

Lemma 17. *Let ub be the upper bound on the weighted treewidth obtained from applying WRATIO (version 1 or version 2) to a graph $G = (V, E, w)$. Then $wtw(G) \leq ub$.*

We end this subsection with some general observations.

Lemma 18. *Let G be a complete weighted graph, $wtw(G) = \prod_{v \in V}(w(v))$.*

Lemma 19. *The weighted treewidth of a weighted triangulated graph G equals the upper bound obtained from the upper bound obtained from the MMNW_ub heuristic.*

Lemma 20. *The weighted treewidth of a weighted triangulated graph G equals the upper bound obtained from the following upper bound heuristics: MF_W heuristic, WMFEO heuristic, WRATIO heuristic.*

3.3 Improving Upper and Lower Bound Heuristics

In order to obtain better lower and upper bounds on the weighted treewidth from the above heuristics, it is wisely to incorporate some safe rules in the *selection conditions* of these heuristics. One can do that as follows. At each step when we have to select and eliminate a vertex for the graph using some specific selection conditions, we choose the vertices that do not cause a worse upper or lower bound than that we will obtain when we select other vertices. An example for such safe rules is the rule of selecting simplicial and strongly almost simplicial vertices whenever they exist. The question that could be arise now is the following. Is it worthwhile to spent some time to test, at each state, whether or not there are simplicial or strongly almost simplicial vertices in the graph? Theoretically, it has been proved that selecting of these vertices is safe (see Lemma's 12 and 13). Practically, the results of our experiments reported in Section 4 and the results reported in [8] support this.

3.4 A Branch and Bound Algorithm for Weighted Treewidth

After we have introduced a number of heuristics for determining lower and upper bounds on weighted treewidth in the previous sections, we introduce in this section a weighted variant of branch and bound algorithm, *BB-tw* that we have introduced in [3], for computing the exact weighted treewidth of weighted graphs. The goals for developing this algorithm were: First, to determine the exact weighted treewidth of some graphs, in particular for graphs with at most 50 vertices. Second, to be able to more precisely establish the quality of the given upper and lower bound, as an exact algorithm allows us to compare the outcome of these heuristics with the exact values.

Third, we can use branch and bound algorithm for improving the upper and lower bounds on the weighted treewidth, obtained from the above heuristics, as we have described in [3], if determining the exact weighted treewidth is not possible within a reasonable time.

In the weighted variant of branch and bound algorithm, W-BB-tw, we have the same space of all feasible solutions as that we have described for unweighted variant. Briefly, this space consists of all possible elimination orderings of the vertices of the given graph. The input to the algorithm are a weighted graph $G = (V, E, w)$, the best known upper and lower bounds, obtained from the heuristics for the weighted treewidth of G described in this paper, and a perfect elimination ordering that gave the best upper bound that is known. The algorithm works as follows: At the beginning, we check whether the best upper bound, ub, equals the best lower bound, lb, obtained from the upper and lower bound heuristics. If so, then the algorithm return this value as the exact weighted treewidth of G. Otherwise, we test every (apart from pruning) possible elimination orderings, in the space of all feasible solutions, whether the elimination of the vertices of the graph due to this order produces an exact weighted treewidth or a better upper bound than reported so far. Moreover, we prune any solution in that space, which delivers an upper bound that is greater than or equal to the reported one so far. The steps for eliminating the vertices of the graph and producing a triangulation of G for each elimination ordering are as it is described in Figure 1.

The pruning rules that we have incorporated in this variant of the algorithm are similar to those we used for unweighted variant in [3]. We have adapted all the pruning

rules we have described for unweighted variant to be used for the weighted variant. The following two Lemmas show the differences in the methods of computing the treewidth of a graph and the weighted treewidth of a weighted graph.

Lemma 21. *Let $H_1, \cdots, H_r$, $r \geq 1$, be all possible triangulated graphs of a graph G. Let $Q_1, \ldots, Q_r$ be the cliques of maximum size in $H_1, \cdots, H_r$. Then, the treewidth of G equals the minimum size of $Q_1, \cdots, Q_r$.*

Lemma 22. *Let $H_1, \cdots, H_r$, $r \geq 1$, be all possible triangulated graphs of a weighted graph G. Let $Q_1, \ldots, Q_r$ be the cliques of maximum weight in $H_1, \cdots, H_r$. Then, the weighted treewidth of G equals the minimum weight of $Q_1, \cdots, Q_r$.*

The differences in the manners in which the pruning rules can be used in both variants of the problem follow from the differences in these two characterizations of treewidth and weighted treewidth. Consider the pruning rules given in Section 3.2 of [3]. Some of these pruning rules have to be modified when we consider the weighted variant, while the others remain as in the unweighted variant. Below, we discuss the rules that are modified for the weighted case. The rules that are not changed can be found in [3].

Pruning Rule 2: The Weight of the Temporary Graph. Let G' be the temporary graph obtained from eliminating a set of vertices X from a given weighted graph G. Let max be the maximum neighborhood weight of all vertices $x \in X$, at the step when they were eliminated from G and added to X. If the total weight of the vertices in G' is less than or equal to the value of max, then we replace the upper bound value reported so far with the value of max, and prune the subtree rooted at the last vertex, $x \in X$, that has been eliminated from G and added to the set X, from the space of all feasible solutions.

Lemma 23. *Let $G' = (W, F)$ be the graph obtained from eliminating a vertex y from a weighted graph $G = (V, E)$, let $r = nw(y)$ at the step when y is eliminated from G. If $\prod_{w \in W} w(w) < r$, then the treewidth of G is at most r.*

Pruning Rule 3: The Weight of the Eliminated Vertex. We check in this rule whether the neighborhood weight of the vertex that we have to eliminate, $nw(v)$, is greater than or equal to the best upper bound on the weighted treewidth reported so far. If such a case holds, then the current elimination ordering will not generate a better upper bound on the weighted treewidth than the one we have reported right now. Therefore, we prune this elimination ordering from the space of all feasible solutions and continue the search operation for the exact weighted treewidth or a better upper bound in the next elimination ordering.

Lemma 24. *Let $H = (W, F, w)$ be a triangulation of a weighted graph $G = (V, E, w)$ and ub be an upper bound on the weighted treewidth of G. If $\exists w \in W$, $nw(w) > ub$ and w is simplicial, then $wtw(G) < wtw(H)$.*

Pruning Rule 5: Simplicial and Strongly Almost Simplicial Vertices. In the Pruning Rule 5 of the branch and bound algorithm BB-tw for unweighted graphs introduced in [3], if the graph contains any simplicial or a strongly almost simplicial with a degree at

most the best upper bound reported so far, then we eliminate this vertex or these vertices from the graph and prune all the elimination orderings from the space of all feasible solutions which, the values of their elements equal the values of their correspond elements of the current elimination ordering up to this position. In the weighted variant of the algorithm, this rule is a rather straightforward generalization of the unweighted variant in the case of simplicial vertices. But, in the case of almost simplicial vertices, a vertex should fulfills to the following conditions to be strongly almost simplicial: The vertices in its neighborhood form an almost clique, its neighborhood weight is at most the best upper bound reported right now, and its weight is at most the weight of the excluding vertex from its neighbor.

4 Computational Experiments

We now briefly report on computational experiments on our upper and lower bound heuristics and the exact algorithm. The algorithms were tested on graphs from real world applications. The heuristics appear to be very fast. The MFEOF heuristic appears to usually very well. The Ratio-2 heuristic is always outperformed by the other heuristics. Preprocessing appears to be very useful tool, which is illustrated by the fact that MMNW_lb with preprocessing gives better bounds than γ_w without. In several cases, lower and upper bounds match, and we have the exact weighted treewidth. In other cases, there is a big gap. The branch and bound algorithm works well for exactly computing the weighted treewidth for graphs with up to fifty vertices. For more details, see [4].

Acknowledgments. We would like to thank Arie Koster for valuable suggestions and helpful comments.

References

1. Arnborg, S., Corneil, D.G., Proskurowski, A.: Complexity of finding embeddings in a k-tree. SIAM J. Alg. Disc. Meth. 8, 277–284 (1987)
2. Bachoore, E.H., Bodlaender, H.L.: New upper bound heuristics for treewidth. In: Nikoletseas, S.E. (ed.) WEA 2005. LNCS, vol. 3503, pp. 217–227. Springer, Heidelberg (2005)
3. Bachoore, E.H., Bodlaender, H.L.: A branch and bound algorithm for exact, upper, and lower bounds on treewidth. In: Cheng, S.-W., Poon, C.K. (eds.) AAIM 2006. LNCS, vol. 4041, pp. 255–266. Springer, Heidelberg (2006)
4. Bachoore, E.H., Bodlaender, H.L.: Weighted treewidth. algorithmic techniques and results. Technical Report UU-CS-2006-012, Department of Information and Computing Sciences, Utrecht University, Utrecht, The Netherlands (2006) Available at `http://www.cs.uu.nl/people/bachoore`
5. Bodlaender, H.L.: A partial k-arboretum of graphs with bounded treewidth. Theor. Comp. Sc. 209, 1–45 (1998)
6. Bodlaender, H.L., Koster, A.M.C.A., van den Eijkhof, F.: Pre-processing rules for triangulation of probabilistic networks. Computational Intelligence 21(3), 286–305 (2005)
7. van den Eijkhof, F., Bodlaender, H.L.: Safe reduction rules for weighted treewidth. In: Kučera, L. (ed.) WG 2002. LNCS, vol. 2573, pp. 176–185. Springer, Heidelberg (2002)

8. van den Eijkhof, F., Bodlaender, H.L., Koster, A.M.C.A.: Safe reduction rules for weighted treewidth. Algorithmica (to appear)
9. Jensen, F.V.: Bayesian Networks and Decision Graphs. Statistics for Engineering and Information Science. Springer, New York (2001)
10. Lauritzen, S.J., Spiegelhalter, D.J.: Local computations with probabilities on graphical structures and their application to expert systems. The Journal of the Royal Statistical Society. Series B (Methodological) 50, 157–224 (1988)
11. Pearl, J.: Probablistic Reasoning in Intelligent Systems: Networks of Plausible Inference. Morgan Kaufmann, Palo Alto (1988)
12. Ramachandramurthi, S.: The structure and number of obstructions to treewidth. SIAM J. Disc. Math. 10, 146–157 (1997)

Spanning Trees with Many Leaves in Regular Bipartite Graphs

Emanuele G. Fusco and Angelo Monti

Dipartimento di Informatica
"Sapienza", Università di Roma , via Salaria, 113-00198 Rome, Italy
{fusco,monti}@di.uniroma1.it

Abstract. Given a d-regular bipartite graph G_d, whose nodes are divided in *black* nodes and *white* nodes according to the partition, we consider the problem of computing the spanning tree of G_d with the maximum number of black leaves. We prove that the problem is NP hard for any fixed $d \geq 4$ and we present a simple greedy algorithm that gives a constant approximation ratio for the problem. More precisely our algorithm can be used to get in linear time an approximation ratio of $2 - 2/(d-1)^2$ for $d \geq 4$. When applied to cubic bipartite graphs the algorithm only achieves a 2-approximation ratio. Hence we introduce a local optimization step that allows us to improve the approximation ratio for cubic bipartite graphs to 1.5.

Focusing on structural properties, the analysis of our algorithm proves a lower bound on $l_B(n,d)$, i.e., the minimum m such that every G_d with n black nodes has a spanning tree with at least m black leaves. In particular, for $d = 3$ we prove that $l_B(n,3)$ is exactly $\lceil \frac{n}{3} \rceil + 1$.

1 Introduction

The problem of finding spanning trees with many leaves has been thoroughly investigated [1,2,4,5,6,8,9,10,11,13,14,15,17]. It is known to be NP hard [4]. Lu and Ravi [14,15] provided 3-approximation algorithms and a 2-approximation algorithm was presented by Solis-Oba [17]. It is known that the problem remains NP hard even if the input is restricted to d-regular graphs for any fixed $d \geq 3$ [11]. A 7/4 approximation algorithm for cubic graphs is presented in [13]. Finding approximation algorithms with ratio less than 2 for d-regular graphs remains an open problem for $d \geq 4$.

The NP hardness of the optimization problem leads to seek constructive proofs for related extremal problems. A constructive proof that all graphs in a particular class have spanning trees with at least m leaves becomes an algorithm to produce such a tree for graphs in this class. Let $l(n,d)$ be the maximum integer m such that every connected n-vertex graph with minimum vertex degree at least d has a spanning tree with at least m leaves. The value $l(n,d)$ is known for $d \leq 5$. Trivially $l(n,2) = 2$. Storer [18] proved that $l(n,3) = \lceil n/4 + 2 \rceil$. Griggs and Wu [9] and Kleitman and West [10] proved $l(n,4) = \lceil \frac{2}{5}n + \frac{8}{5} \rceil$. In [9] it is also

T. Tokuyama (Ed.): ISAAC 2007, LNCS 4835, pp. 904–914, 2007.

proved that $l(n,5) = \lceil \frac{3}{6}n + 2 \rceil$. For $d \geq 6$ the exact value of $l(n,d)$ remains unknown. For more information about this topic see [3].

In [16] a variation of the maximum leaf spanning tree problem has been introduced. This variation restricts the problem to bipartite graphs and asks to find a spanning tree having the maximum number of leaves in one of the partited set. We call nodes in this set *black nodes*, and *white nodes* those in the other. In [12] it is proved that this variation of the problem is NP hard for planar bipartite graphs. In this paper we study the variation of the problem proposed in [16] restricted to the class of regular bipartite graphs. We prove that the problem is NP hard for d-regular bipartite graphs for any fixed $d \geq 4$. We remark that our proof of NP hardness relies on a construction involving *non planar* regular bipartite graphs. It remains an open question to determine if the problem is NP hard for regular planar bipartite graphs.

We present greedy algorithms working in linear time that find, for any d-regular graph, a spanning tree having a constant fraction of the maximum number of black leaves. Our algorithm for d-regular bipartite graphs provides an approximation ratio of $2 - 2/(d-1)^2$. The analysis of the performance ratio is based on the assumption that $d \geq 4$: in order to reach approximation ratio 1.5 on cubic bipartite graphs we present a refinement on our base algorithm based on local optimization.

Define $l_B(n,d)$ as the maximum m such that every d-regular bipartite graph with n black nodes has a spanning tree with at least m black leaves. Trivially $l_B(n,2) = 1$. We prove that $l_B(n,3) = \lceil n/3 \rceil + 1$. For $d \geq 4$ the exact value of $l_B(n,d)$ remains unknown, however we provide upper and lower bounds. More precisely we prove: $\left\lceil \frac{d-1}{2d}n + \frac{(d-1)^2}{2d} \right\rceil \leq l_B(n,d) \leq \left\lceil \frac{d-2}{d}n \right\rceil + 1$.

2 Preliminaries

In this section we introduce some terminology and notation. Let G be a graph; we use $V(G)$ to denote the set of nodes in G and $E(G)$ to denote the set of edges in G. For a node v in $V(G)$, $\Gamma_G(v)$ denotes the set of neighbours of v in G. We denote by G_d a d-regular bipartite graph. We use colors *black* and *white* to identify the two sets of the partition. As in any regular bipartite graph the number of black nodes is equal to the number of white nodes, we use the letter n to denote the number of black nodes in G_d (clearly, the total number of nodes in G_d is equal to $2n$).

Given a spanning tree T of G_d, $\lambda_i(T)$, $1 \leq i \leq d$, denotes the number of black nodes of degree i in T. We omit T when it is clear form the context.

Lemma 1. *Let T be a spanning tree of G_d, it holds*

$$\lambda_1(T) = 1 + \sum_{i=3}^{d} (i-2)\lambda_i(T) \tag{1}$$

Proof. As T spans all the nodes in G_d, it holds that $\sum_{i=1}^{d} \lambda_i = n$. Any edge in T has endpoints in one black node and one white node, so the total number of

edges is given by the sum of the degree of the black (or white) nodes, giving that $\sum_{i=1}^{d} i\lambda_i = 2n - 1$. From these two equations we obtain Equation 1. □

3 Regular Bipartite Graphs

We start our analysis from the general case of regular bipartite graphs, while on Sec. 4 we focus on cubic graphs to refine our results in this restricted case.

Lemma 2. *Let T be a spanning tree of G_d, then $\lambda_1(T) \leq \left\lfloor \frac{(d-2)n+1}{d-1} \right\rfloor$.*

Proof. From Equation 1, as $\sum_{i=1}^{d} \lambda_i = n$, we have that λ_1 is maximized when $\lambda_1 + \lambda_d = n$ and $\lambda_2, \lambda_3, \ldots, \lambda_{d-1}$ are all 0. As we search for integer solutions, $\lambda_1 \leq \frac{(d-2)n+1}{d-1}$ and the thesis follows. □

We now describe the algorithm *span* that, given a graph G_d, produces a spanning tree T_A for G_d. The algorithm first builds a forest $\mathcal{F}$, then it connects the trees in $\mathcal{F}$ and the isolated nodes to form T_A. Every tree T_i in $\mathcal{F}$ is built by first choosing a black node v such that $\Gamma_{G_d}(v) \cap V(\mathcal{F}) = \emptyset$. Each tree T_i is augmented as long as a new black node w with at most $d-2$ neighbours in T_i can be found. When a tree T_i cannot be augmented, the algorithm starts building a new tree T_{i+1}.

In Algorithm 1 we formalize the algorithm *span* by providing its pseudo code.

Algorithm 1. span(G_d)

```
1: F ← ∅
2: i ← 1
3: while ∃ a black node v ∈ G_d \ V(F) such that Γ_{G_d}(v) ∩ V(F) = ∅ do
4:    E(T_i) ← {(v, x) such that x ∈ Γ_{G_d}(v)}
5:    V(T_i) ← {v} ∪ Γ_{G_d}(v)
6:    while ∃ a black node w ∈ G_d \ (V(F) ∪ V(T_i)) and a white node y ∈
      V(T_i) such that |Γ_{G_d}(w) ∩ (V(F) ∪ V(T_i))| ≤ d − 2 and (w, y) ∈ E(G_d)
      do
7:       E(T_i) ← E(T_i) ∪ {(w, y)} ∪ {(w, z) such that z ∈ Γ_{G_d}(w) \ Γ_{T_i}(w)}
8:       V(T_i) ← V(T_i) ∪ {w} ∪ Γ_{G_d}(w)
9:    F ← F ∪ T_i
10:   i ← i + 1
11: build T_A by connecting F and all the nodes in V(G_d) − V(F) in a tree
12: return T_A
```

Lemma 3. *For any G_d with $d \geq 4$ there exists a spanning tree T_A such that*

$$\lambda_1(T_A) \geq \left\lceil \frac{d-1}{2d} n + \frac{(d-1)^2}{2d} \right\rceil$$

Proof. The proof of this lemma consists in the performance evaluation of Algorithm 1.

Let $T_1, T_2, \ldots, T_k$ be the trees built by the algorithm with input G_d. Any black node in a tree T_i has degree at least 3 and all the black nodes that do not belong to any tree have at least $d-1$ neighbours belonging to one tree T_h for some value $1 \leq h \leq k$. Let $\overline{b}_i$ be the number of black nodes in $V(G_d) \setminus V(\mathcal{F})$ having at least $d-1$ neighbours in T_i. Obviously,

$$n = \sum_{i=1}^{k} \left(\overline{b}_i + \sum_{j=3}^{d} \lambda_j(T_i) \right) \tag{2}$$

Now we bound the number $\overline{b}_i$ with $\left\lfloor \frac{d\left(1+\sum_{j=3}^{d}(j-2)\lambda_j(T_i)\right)}{d-1} \right\rfloor$ by considering that T_i contains $1+\sum_{j=3}^{d}(j-1)\lambda_j(T_i)$ white nodes and hence there are $d(1+\sum_{j=3}^{d}(j-2)\lambda_j(T_i))$ edges from nodes in $V(T_i)$ to nodes in $V(G_d) \setminus V(T_i)$.

From Equation 2 we have $n \leq \frac{dk+\sum_{j=3}^{d}(dj-d-1)\lambda_j(T_A)}{d-1}$ and as in the forest each tree T_i has at least one node of degree d (i.e. the root) we have that

$$n \leq \frac{d\lambda_d(T_A) + \sum_{j=3}^{d}(dj-d-1)\lambda_j(T_A)}{d-1} \tag{3}$$

If $d \geq 4$, $d^2-4d+1 > 0$, and as $\lambda_d(T_A) \geq 1$ the following chain of disequations holds:

$$\begin{aligned}
d\lambda_d(T_A) + \sum_{j=3}^{d} (dj-d-1)\,\lambda_j(T_A) = \\
d\lambda_d(T_A) + \sum_{j=3}^{d} 2d\,(j-2)\,\lambda_j(T_A) - \sum_{j=3}^{d} ((j-3)d+1)\,\lambda_j(T_A) \leq \\
d\lambda_d(T_A) + 2d\,(\lambda_1(T_A)-1) - ((d-3)d+1)\,\lambda_d(T_A) = \\
2d\lambda_1(T_A) - 2d - \lambda_d(T_A)(d^2-4d+1) \leq \\
2d\lambda_1(T_A) - 2d - (d^2-4d+1) = \\
2d\lambda_1(T_A) - (d-1)^2
\end{aligned}$$

Hence from Equation 3 we have

$$n \leq \frac{2d\lambda_1(T_A) - (d-1)^2}{d-1}$$

and the thesis follows. □

Combining Lemma 3 and Lemma 2 we can state the following theorem:

Theorem 1. *The problem of finding a Spanning Tree with the maximum number of black leaves for a d-regular bipartite graph G_d, $d \geq 4$, can be approximated by an algorithm running in linear time with approximation ratio $\leq 2 - 2/(d-1)^2$.*

Proof. Let T^* be a spanning tree of G_d with the maximum number of black leaves and let T_A be the spanning tree of G_d produced by Algorithm 1. From Lemma 2 we have that $\lambda_1(T^*) \leq \left\lfloor \frac{(d-2)n+1}{d-1} \right\rfloor$ and from Lemma 3 we have $\lambda_1(T_A) \geq \frac{d-1}{2d}n + \frac{(d-1)^2}{2d}$. It follows that $\frac{\lambda_1(T^*)}{\lambda_1(T_A)} \leq 2\left(\frac{d(d-2)n+d}{(d-1)^2n+(d-1)^3}\right) < 2\frac{d(d-2)}{(d-1)^2} = 2 - \frac{2}{(d-1)^2}$. □

4 Cubic Bipartite Graphs

Algorithm 1 can obviously be applied to a graph G_3. An analysis for the case $d = 3$ gives $\lambda_1(T_A) \geq n/4$ and combining this with Lemma 2 we obtain an approximation ratio of 2.

Now consider the example in Figure 1: the graph in the example is a necklace composed by l repetitions of the same block. A run of Algorithm 1 can produce a forest where every tree T_i contains a black node only (thick edges in figure represent the edges in the forest). In such a case, $\lambda_3 = l$ and so $\lambda_1 = l + 1$, while the optimum solution can assign degree 3 to all but one the black nodes on the bottom, thus achieving $2l$ black leaves.

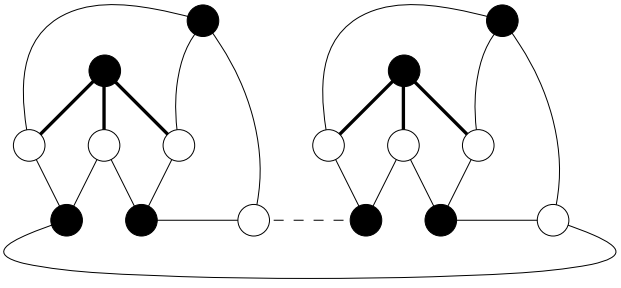


Fig. 1. Worst case example for Algorithm 1 on cubic graphs

As suggested by the example above, in order to improve the approximation ratio, we modify Algorithm 1 by adding a procedure that tries to reduce the number of trees with only one black node in $\mathcal{F}$. We call *span*$_3$ the modified version of Algorithm 1. If a tree T_i has only one black node at the end of the while loop of line 6 of Algorithm 1, span$_3$ searches in $G_d \setminus \mathcal{F}$ for the pattern depicted in figure 2.a. If such a pattern is present, T_i is destroyed and rebuilt starting from node x. Then, the augmenting process of lines 6 to 8 of the original algorithm is applied, resulting in a tree with at least 2 black nodes.

Lemma 4. *For any G_3 there exists a spanning tree T_A such that $\lambda_1(T_A) \geq \lceil \frac{n}{3} \rceil + 1$.*

Proof. The proof of this lemma consists in the performance evaluation of Algorithm span$_3$. In the following we assume G_3 has at least 5 black nodes (if $n = 3$ or $n = 4$, the lemma trivially holds as any spanning tree having one black node

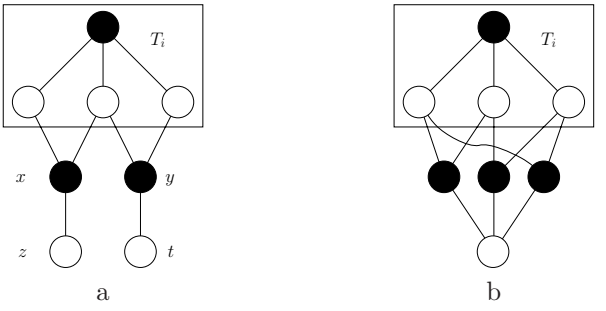


Fig. 2. a) Pattern for local optimization in Algorithm span$_3$. b) Performance analysis case.

of degree 3 and 2 black leaves is optimal). Let $T_1, T_2, \ldots, T_k$ be the trees built by Algorithm span$_3$ with input G_3. All black nodes in $V(\mathcal{F})$ have degree 3 while black nodes in $V(G_3) \setminus V(\mathcal{F})$ have 2 neighbors in some tree $T_i \in \mathcal{F}$. Let $\overline{b}_i$ be the number of black nodes outside $\mathcal{F}$ that have at least 2 neighbors in the tree $T_i \in \mathcal{F}$. It holds that

$$n = \sum_{i=1}^{k} \left(\overline{b}_i + \lambda_3(T_i)\right)$$

Now we prove that $\overline{b}_i \leq 2\lambda_3(T_i)$ for all $1 \leq i \leq k$: notice that this is enough to prove the lemma since it implies that $n \leq 3\lambda_3(T_A) = 3\lambda_1(T_A) - 3$.

With the same reasoning applied in Lemma 3, we can bound $\overline{b}_i$ with $\left\lfloor \frac{3\lambda_3(T_i)+3}{2} \right\rfloor$. If $\lambda_3(T_i) \geq 2$ it holds $\overline{b}_i \leq 2\lambda_3(T_i)$ so the only case we need to analyze is when $\lambda_3(T_i) = 1$.

Assume by contradiction that $\overline{b}_1 = 3$: there must be 3 black nodes, say $x, y,$ and z, such that each of them has two white neighbors in T_i. Now consider the set W of white neighbors of $x, y,$ and z outside T_i. It cannot be $|W| \geq 2$ as otherwise the pattern of figure 2.a would have been detected by span $_3$ and T_i would have been a tree with at least 2 black nodes. On the other hand, it cannot be $|W| = 1$ as the only G_3 where this happens is the one depicted in figure 2.b: this graph has 4 black nodes only while we assumed $n \geq 5$. It follows that $\overline{b}_i \leq 2$ thus completing the proof. □

Remark 1. The lower bound given in Lemma 4 is tight. For a given value n we can build a graph composed by $l = \lfloor n/3 \rfloor$ subgraphs closed on a necklace. The fist $l - 1$ subgraphs are repetitions of $k_{3,3} - e$ while the last one can have $3, 4$ or 5 black and white nodes depending on the value of n mod 3 (see figure 3 for an example where $n \bmod 3 = 0$).

Any spanning tree has to assign degree greater than 1 to at least 2 black nodes in any subgraph but one. It follows that $\lambda_1 \leq n - 2\lfloor n/3 \rfloor + 1 = \lceil n/3 \rceil + 1$.

Theorem 2 below immediately follows from Lemma 2 and Lemma 4.

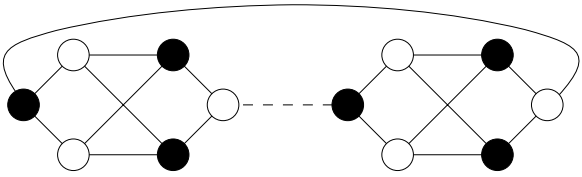


Fig. 3. G_3^e

Theorem 2. *The problem of finding a Spanning Tree with the maximum number of black leaves for a cubic bipartite graph G_3 can be approximated by an algorithm running in linear time with approximation ratio $\leq 3/2$.*

Remark 2. The analysis of Algorithm span$_3$ is tight. Consider the example in figure 4: the graph in the example is a necklace composed by l repetitions of the same block. A run of algorithm span$_3$ can produce a forest where every tree T_i contains two black nodes only (thick edges in figure represent the edges in the forest). In such a case, $\lambda_3 = 2l$ and so $\lambda_1 = 2l + 1$, while the optimum solution can assign degree 3 to all but one the black nodes on the bottom, thus achieving $3l$ black leaves.

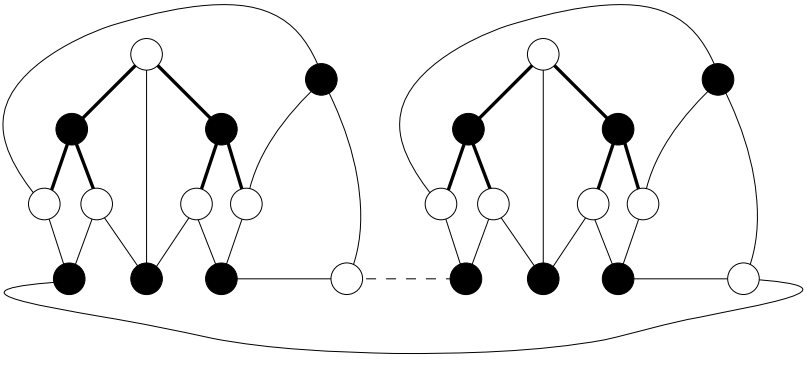


Fig. 4. Worst case example for Algorithm span$_3$

5 NP Hardness

In this section we prove that the problem of finding a spanning tree with the maximum number of black leaves in a 4-regular bipartite graph is NP-hard. Details of the extension of the proof for any fixed $d \geq 4$ are omitted. Our proof relies in a reduction to a restricted version of the well known NP-complete problem 3-*exact cover*. 3-exact cover (in short 3EC) requires, given a universe $\mathcal{U}$ and a collection $\mathcal{S}$ of 3-subsets of $\mathcal{U}$, to determine if there exists a subcollection $\mathcal{S}'$ of pairwise disjoint sets in $\mathcal{S}$ that forms a partition of $\mathcal{U}$.

We consider instances of 3EC where each element of $\mathcal{U}$ occurs in *exactly* three subsets of $\mathcal{S}$ and $|\mathcal{U}| = 4 \cdot 3^i$, $i \geq 1$. We denote 3EC* this restricted version of

3EC. It is known that 3EC remains NP-complete when each element of $\mathcal{U}$ occurs in *at most* 3 subsets of $\mathcal{S}$ (see, e.g., [7] pag. 222). From this fact it is a simple exercise to prove that 3EC is polynomially reducible to 3EC*. Hence we have:

Lemma 5. *3EC* is NP complete.*

Theorem 3. *The problem of determining, given G_4, if there exists a spanning tree T of G_4 such that $\lambda_2(T) = \lambda_3(T) = 0$ is NP complete.*

Proof. Starting from any instance I of 3EC*, we construct in polynomial time a graph G_4, and the thesis follows from the fact that G_4 admits a spanning tree T with $\lambda_2(T) = \lambda_3(T) = 0$ if and only if I admits a solution.

To build G_4, we create a black node for each set in $\mathcal{S}$. We add new nodes to form a tree whose leaves are the $4 \cdot 3^i$ black nodes representing sets in $\mathcal{S}$. The tree is made by white nodes of degree 4 and internal black nodes of degree 2. More precisely the tree is rooted in a white node; even levels contain white nodes while odd levels contain black nodes. By construction, the tree will have $2i + 1$ levels and at level $2j + 1$, $j \geq 0$ there will be $4 \cdot 3^j$ black nodes (i.e., the levels with black nodes have an even number of nodes). Then we create a white node for each element of the universe $\mathcal{U}$ and we put an edge between the white node representing element $u_i \in \mathcal{U}$ and the black leaf representing set $S_j \in \mathcal{S}$ if $u_i \in S_j$.

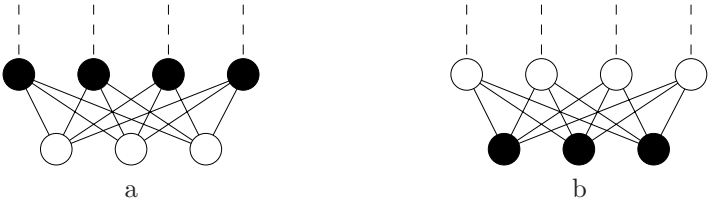


Fig. 5. a) Gadget *Ga1*. b) Gadget *Ga2*.

To complete the construction, we connect each of the white nodes representing elements in $\mathcal{U}$ to the outgoing edges of a gadget Ga1 (see figure 5.a) and we connect each of the black nodes with degree 2 with a gadget Ga2 twice (see figure 5.b): this operation can always be done as the number of internal black nodes in the tree is even. The resulting graph is outlined in figure 6.

Given a solution for I, it is easy to derive a spanning tree for G_4, having black nodes of degree 4 and 1 only, by assigning degree 4 to all the black nodes connected to gadgets Ga2 and to the black nodes representing sets forming the solution for I. On the other hand, if a spanning tree T^* of G_4 exists having black nodes of degree 4 and 1 only, a solution for I can always be found. First notice that in any such T^*, all the black nodes connected to gadgets Ga2 must have degree 4, as nodes inside the gadgets must be connected to the tree and black nodes inside the gadgets cannot be assigned degree 4 without resulting in some black node of degree 2 or 3. As a result, all the black nodes representing sets of

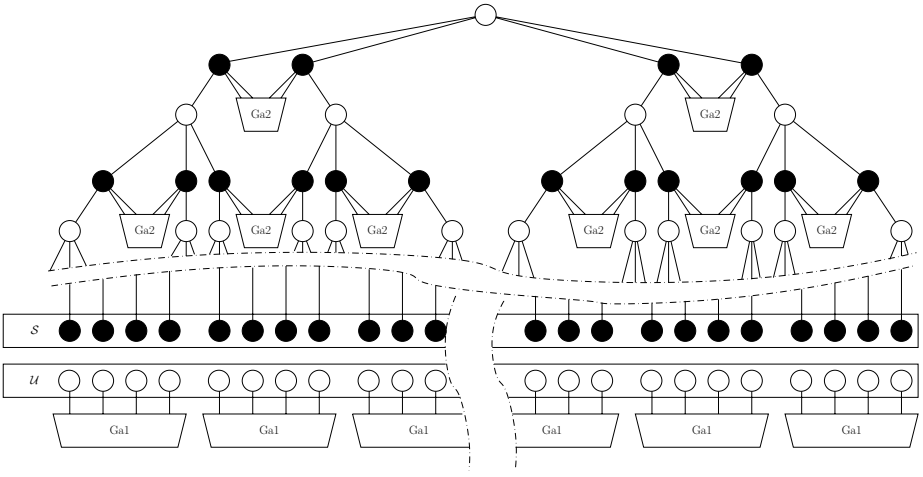


Fig. 6. Construction of graph for the reduction. Edges between black nodes representing sets in $\mathcal{S}$ and white nodes representing elements of $\mathcal{U}$ are missing.

$\mathcal{S}$ in G_4 that have degree 4 in T^* are connected from above, thus they cannot be adjacent in T^* to the same white node representing an element of $\mathcal{U}$ without creating a loop (i.e., if we consider the sets represented by black nodes of degree 4 in T^*, they are pairwise disjoints). Finally, all the white nodes representing elements in $\mathcal{U}$ need to be connected from above in T^*, as connecting a white node from below means passing trough a gadget Ga1, thus producing at least a black node of degree 2 or two of degree 3. It follows that starting from T^*, a solution for I can be found by taking all sets of S corresponding to black nodes of degree 4 in T^*. □

Corollary 1. *The problem of finding a spanning tree with the maximum number of black leaves is NP hard for* 4*-regular bipartite graphs.*

Proof. Define the BLST problem as follows: given G_d and an integer k, determine if it exists a spanning tree T of G_d such that $\lambda_1(T) \geq k$. It is easy to show that BLST on 4-regular bipartite graphs is NP complete. Indeed, from Theorem 3, we know that it is NP complete to determine if a given G_4 has a spanning tree T such that $\lambda_2(T) = \lambda_3(T) = 0$. It follows that the BLST problem is NP complete when $k = \frac{(d-2)n+1}{d-1}$. Having an algorithm that resolves in polynomial time the problem of finding a spanning tree of a given G_4 with the maximum number of black leaves, would allow us to solve BLST in polynomial time, thus proving the problem to be NP hard. □

All the proofs in this section can be extended to any value $d \geq 5$. We omit the details.

Theorem 4. *The problem of finding a spanning tree with the maximum number of black leaves is NP hard for* d*-regular bipartite graphs for any fixed* $d \geq 4$*.*

6 Conclusions and Open Questions

Spanning trees of connected graphs are a major topic of research in the area of graph algorithms. In this paper we studied the problem of finding a spanning tree with the maximum number of black leaves in regular bipartite graphs.

We proved that the problem is NP hard for any fixed $d \geq 4$ and we presented a linear time algorithm that achieve approximation ratio $2 - 2/(d-1)^2$.

It is an interesting question whether this problem is NP hard or polynomial time solvable in the case of cubic bipartite graphs. Our contribution for this class of graphs is a linear time algorithm with approximation ratio 1.5.

Our proof of NP hardness relies on a construction involving regular bipartite graphs that are non planar. It remains an open question to determine if the problem remains NP hard for regular planar graphs also. In [12] it is shown that the problem is NP hard for planar *non regular* bipartite graphs.

Finally, define $l_B(n,d)$ to be the maximum m such that every G_d with n black nodes has a spanning tree with at least m black leaves. Obviously $l_B(n,2) = 1$. From Lemma 4 and Remark 1 we have $l(n,3) = \lceil \frac{n}{3} \rceil + 1$. It would be interesting to determine precisely the value $l_B(n,d)$ for any $d \geq 4$. We know that

$$\left\lceil \frac{d-1}{2d} n + \frac{(d-1)^2}{2d} \right\rceil \leq l_B(n,d) \leq \left\lceil \frac{d-2}{d} n \right\rceil + 1$$

where the lower bound follows from Lemma 4 and the upper bound can be obtained by generalizing Remark 1, using as a building block for the necklace the graph $k_{d,d} - e$.

References

1. Bodlaender, H.L.: On linear time minor tests and depth first search. In: Dehne, F., Santoro, N., Sack, J.-R. (eds.) WADS 1989. LNCS, vol. 382, pp. 577–590. Springer, Heidelberg (1989)
2. Bonsma, P.S., Brüggemann, T., Woeginger, G.J.: A faster fpt algorithm for finding spanning trees with many leaves. In: Rovan, B., Vojtáš, P. (eds.) MFCS 2003. LNCS, vol. 2747, pp. 259–268. Springer, Heidelberg (2003)
3. Caro, Y., West, D.B., Yuster, R.: Connected domination and spanning trees with many leaves. SIAM Journal on Discrete Mathematics 13(2), 202–211 (2000)
4. Ding, G., Johnson, T., Seymour, P.: Spanning trees with many leaves. Journal of Graph Theory 37, 189–197 (2001)
5. Fujie, T.: The maximum-leaf spanning tree problem: Formulations and facets. Networks 43(4), 212–223 (2004)
6. Galbiati, G., Maffioli, F., Morzenti, A.: A short note on the approximability of the maximum leaves spanning tree problem. Inf. Process. Lett. 52(1), 45–49 (1994)
7. Garey, M.R., Johnson, D.S.: Computers and Intractability; A Guide to the Theory of NP-Completeness. W. H. Freeman & Co., New York (1979)
8. Griggs, J.R., Kleitman, D.J., Shastri, A.: Spanning trees with many leaves in cubic graphs. Journal of Graph Theory 13(6), 669–695 (1989)

9. Griggs, J.R., Wu, M.: Spanning trees in graphs of minimum degree 4 or 5. Discrete Math. 104(2), 167–183 (1992)
10. Kleitman, D.J., West, D.B.: Spanning trees with many leaves. SIAM Journal on Discrete Mathematics 4(1), 99–106 (1991)
11. Lemke, P.: The maximum leaf spanning tree problem for cubic graphs is np-complete. Technical Report IMA Preprint Series # 428, University of Minnesota (July 1988)
12. Li, P.C., Toulouse, M.: Variations of the maximum leaf spanning tree problem for bipartite graphs. Inf. Process. Lett. 97(4), 129–132 (2006)
13. Lorys, K., Zwozniak, G.: Approximation algorithm for the maximum leaf spanning tree problem for cubic graphs. In: Möhring, R.H., Raman, R. (eds.) ESA 2002. LNCS, vol. 2461, pp. 686–697. Springer, Heidelberg (2002)
14. Lu, H.-I, Ravi, R.: The power of local optimizations: Approximation algorithms for maximum-leaf spanning tree (draft)*. Technical Report CS-96-05 (1996)
15. Lu, H.-I, Ravi, R.: Approximating maximum leaf spanning trees in almost linear time. J. Algorithms 29(1), 132–141 (1998)
16. Rahman, M.S., Kaykobad, M.: Complexities of some interesting problems on spanning trees. Inf. Process. Lett. 94(2), 93–97 (2005)
17. Solis-Oba, R.: 2-approximation algorithm for finding a spanning tree with maximum number of leaves. In: Bilardi, G., Pietracaprina, A., Italiano, G.F., Pucci, G. (eds.) ESA 1998. LNCS, vol. 1461, pp. 441–452. Springer, Heidelberg (1998)
18. Storer, J.A.: Constructing full spanning trees for cubic graphs. Inf. Process. Lett. 13, 8–11 (1981)

Problem Kernels for NP-Complete Edge Deletion Problems: Split and Related Graphs

Jiong Guo*

Institut für Informatik, Friedrich-Schiller-Universität Jena,
Ernst-Abbe-Platz 2, D-07743 Jena, Germany
guo@minet.uni-jena.de

Abstract. In an edge deletion problem one is asked to delete at most k edges from a given graph such that the resulting graph satisfies a certain property. In this work, we study four NP-complete edge deletion problems where the goal graph has to be a chain, a split, a threshold, or a co-trivially perfect graph, respectively. All these four graph classes are characterized by a common forbidden induced subgraph $2K_2$, that is, an independent pair of edges. We present the seemingly first non-trivial algorithmic results for these four problems, namely, four polynomial-time data reduction algorithms that achieve problem kernels containing $O(k^2)$, $O(k^4)$, $O(k^3)$, and $O(k^3)$ vertices, respectively.

1 Introduction

Given a graph G, a graph property Π (for instance, to belong to a certain graph class), and an integer $k \geq 0$, the Π edge deletion (for short, Π deletion) problem asks for a set of at most k edges whose deletion transforms G into a graph satisfying Π. The solution set is called *Π deletion set*. Edge deletion problems have applications in several areas, such as molecular biology and numerical algebra (see, for example, [2,14,18]), and their computational complexity has been widely studied in the literature. Yannakakis [20] gave the first systematic study of the complexity of edge deletion problems. We refer to [2,14,18] for excellent overviews.

In contrast to the extensive study on the complexity of Π deletion problems, relatively few algorithmic results are known for these problems. A general, constant-factor approximation algorithm was given by Natazon et al. [14] for Π deletion problems on bounded-degree graphs with respect to properties Π that can be characterized by finite sets of forbidden induced subgraphs. A forbidden induced subgraph characterization of a graph property Π means that a graph satisfies Π iff it contains none of a given set $\mathcal{H}$ of graphs as induced subgraphs. Herein, the graphs satisfying Π are also called $\mathcal{H}$-free graphs. Concerning parameterized complexity, Cai [3] showed that, for a graph property Π characterized

* Supported by the Deutsche Forschungsgemeinschaft (DFG), Emmy Noether research group PIAF (fixed-parameter algorithms), NI 369/4, and research project DARE (data reduction and problem kernels), GU 1023/1-1.

T. Tokuyama (Ed.): ISAAC 2007, LNCS 4835, pp. 915–926, 2007.

by a finite set $\mathcal{H}$ of forbidden induced subgraphs, the corresponding Π deletion problem is *fixed-parameter tractable*. More precisely, there exists a search tree based algorithm solving the problem in $O(d^k \cdot p(|G|))$ time with d being the maximum size of the edge sets of the forbidden subgraphs in $\mathcal{H}$ and p a polynomial function of the size of the input graph G.

Problem kernelization has been recognized as one of the most important contributions of fixed-parameter algorithmics to practical computing [5,11,15]. A *kernelization* is a polynomial-time algorithm that transforms a given instance I with parameter k of a problem P into a new instance I' with parameter $k' \leq k$ of P such that the original instance I is a yes-instance with parameter k iff the new instance I' is a yes-instance with parameter k' and $|I'| \leq g(k)$ for a function g. The instance I' is called the *problem kernel*. Recently, kernelizations of Π deletion problems attracted attention of more and more researchers; for example, there is a series of papers improving the problem kernel for Cluster Editing, where the goal graph is required to be a set of disjoint cliques, from quadratic size to linear size [9,17,6,10].

In this work, we will provide kernelization results for several Π deletion problems whose corresponding properties have $2K_2$ as one of their forbidden induced subgraphs. A $2K_2$-graph is an independent pair of edges, that is, a graph with four vertices and two non-adjacent edges. The class of $2K_2$-free graphs contains many important graph classes such as *chain graphs*, *split graphs*, *threshold graphs*, and *co-trivially perfect graphs* [1]. Here, we will study the corresponding Π deletion problems for Π being these four graph classes, denoted as Chain Deletion, Split Deletion, Threshold Deletion, and Co-Trivially Perfect Deletion. The main results are four polynomial-time kernelization algorithms which achieve problem kernels with $O(k^2)$, $O(k^4)$, $O(k^3)$, and $O(k^3)$ vertices for the four problems, respectively. This seem to be the first non-trivial algorithmic results for these problems. Based on the general result by Cai [3] and the *interleaving* technique from [16], our kernelization results imply faster fixed-parameter algorithms for these problems with running times of $O(2^k + mnk)$, $O(5^k + m^4n)$, $O(4^k + kn^4)$, and $O(4^k + kn^4)$, respectively, where n denotes the number of vertices and m denotes the number of edges of a given graph.

2 Preliminaries

Parameterized complexity is a two-dimensional framework for studying the computational complexity of problems [5,7,15]. One dimension is the input size n (as in classical complexity theory) and the other one the *parameter* k (usually a positive integer). A problem is called *fixed-parameter tractable* (fpt) if it can be solved in $f(k) \cdot n^{O(1)}$ time, where f is a computable function only depending on k. A core tool in the development of fixed-parameter algorithms is polynomial-time preprocessing by *data reduction rules*, often yielding a *kernelization*. Herein, the goal is, given any problem instance I with parameter k, to transform it in polynomial time into a new instance I' with parameter k' such that the size of I' is bounded from above by some function only depending on k, $k' \leq k$, and (I, k)

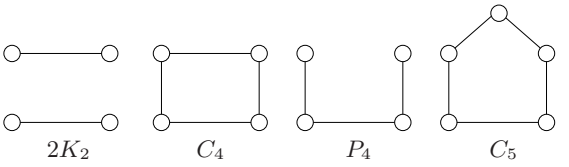


Fig. 1. The forbidden induced subgraphs $2K_2$, C_4, P_4, and C_5

is a yes-instance iff (I', k') is a yes-instance. A data reduction rule is *correct* if the new instance after an application of this rule is a yes-instance iff the original instance is a yes-instance. Throughout this paper, we call a problem instance *reduced* if the corresponding data reduction rules cannot be applied anymore.

We only consider *undirected* graphs $G = (V, E)$, where V is the set of vertices and E is the set of edges. The *complement graph* of G is denoted by $\bar{G} = (V, \bar{E})$ where an edge $\{u, v\} \in \bar{E}$ iff $\{u, v\} \notin E$. The *bipartite complement graph* $\bar{B}$ of a bipartite graph $B = (X, Y, E)$ contains an edge between two vertices $x \in X, y \in Y$ iff $\{x, y\} \notin E$. The *(open) neighborhood* $N_G(v)$ of a vertex $v \in V$ in graph G is the set of vertices that are adjacent to v in G. The *degree* of a vertex v, denoted by $\deg(v)$, is the size of $N_G(v)$. We use $N_G[v]$ to denote the *closed* neighborhood of v in G, that is, $N_G[v] := N_G(v) \cup \{v\}$. For a set of vertices $V' \subseteq V$, the *induced subgraph* $G[V']$ is the subgraph of G over the vertex set V' with the edge set $\{\{v, w\} \in E \mid v, w \in V'\}$. A subset I of vertices is called an *independent set* if $G[I]$ has no edge, whereas a subset K of vertices is called a *clique* if $G[K]$ has all possible edges. For an edge e and an edge set E', we use $G - e$ and $G - E'$ to denote the subgraph of G without e and the edges in E', respectively. For a vertex v and a vertex set V', the notions $G - v$ and $G - V'$ denote the subgraphs of G induced by $V \setminus \{v\}$ and $V \setminus V'$, respectively.

In the following, we will study several graph classes with forbidden induced subgraph characterization. We say that a vertex v *occurs* in a forbidden induced subgraph if v is contained in such an induced subgraph in G. See Fig. 1 for the forbidden induced subgraphs occurring in this work. We call the edge in a P_4 whose both endpoints have degree two the *middle edge*, and call the other two edges the *side edges*.

Compared to the Π vertex deletion problems where one deletes vertices instead of edges, the most difficult point in reducing a given instance of a Π edge deletion problem is how to deal with the vertices that do not occur in any forbidden subgraph. In the vertex version, we can simply remove these vertices, since such vertices can never be included in any optimal solution. However, since deleting an edge could create a new occurrence of a forbidden induced subgraph, it is possible that all optimal solutions of an edge deletion problem have to include an edge not involved in any forbidden subgraph in the original graph. Thus, in this case, we cannot simply remove the vertices that do not occur in any forbidden subgraph. In the following, as one of our main technical contributions, we show that, for the $2K_2$-free graph classes, chain, split, threshold, and co-trivially perfect, such vertices can be removed without affecting the solvability of the corresponding edge deletion problems. We begin with the most simple case, Chain Deletion.

3 Chain Deletion

A bipartite graph $B = (X, Y, E)$ with X and Y being two disjoint vertex subsets is called a *chain graph* if the neighborhoods of the vertices in X form a chain, that is, if there is an ordering of the vertices in X, say x_1, x_2, ..., $x_{|X|}$, such that $N_B(x_1) \subseteq N_B(x_2) \subseteq \ldots \subseteq N_B(x_{|X|})$. It is easy to see that the neighborhoods of the vertices in Y also form a chain. Yannakakis [19] introduced this graph class and proved that the Chain Completion problem, where one is asked whether there is a set of at most k edges whose addition transforms a given bipartite graph into a chain graph, is NP-complete. Since the bipartite complement graph of a chain graph is a chain graph as well, Chain Deletion is NP-complete as well.

The main result of this section consists of two data reduction rules for Chain Deletion that lead to a quadratic-size problem kernel. To this end, we need the following forbidden subgraph characterization of chain graphs given by Yannakakis [19]: A bipartite graph is a chain graph if and only if it does not contain a $2K_2$ as an induced subgraph. Without loss of generality, we assume that the input graph is connected: For a disconnected graph, only some edges of exactly one connected component can be kept. This means that we have to consider each single component individually. We apply the following two data reduction rules to a given bipartite graph $B = (X, Y, E)$:

Rule 1: If there is an edge e involved in more than k $2K_2$'s, then delete e from B, add e to the chain deletion set, and decrease the parameter k by one. If $k < 0$, then report "No".

Rule 2: Delete the vertices from B that do not occur in any $2K_2$.

The correctness of the first rule is easy to verify.

Lemma 1. *Rule 2 is correct.*

Proof. To show the lemma, it suffices to show that, for a vertex v not in any $2K_2$, graph $B = (X, Y, E)$ has a chain deletion set E' with $|E'| \leq k$ if and only if graph $B' := B - v$ has a chain deletion set E'' with $|E''| \leq k$.

"$\Rightarrow$": Since every induced subgraph of a chain graph is a chain graph, this direction is correct.

"$\Leftarrow$": Suppose that E'' is a chain deletion set for B'. Let B'' denote the chain graph resulting by removing the edges in E'' from B'. Then, add v to B'' and connect v to the vertices in $N_B(v)$. By contradiction we show that the resulting graph H is a chain graph and, thus, E'' is a chain deletion set for B. Suppose that H is not a chain graph; this means that v occurs in a $2K_2$ in H.

We associate with the edges in E'' an ordering as follows: Based on the forbidden subgraph $2K_2$, one can easily enumerate all size-at-most-k chain deletion sets for B' by a search tree of height k: At each node α of the search tree, find an induced $2K_2$ and branch into two cases, deleting one edge or the other. Then, recursively treat the two cases. Each of the two search tree edges between node α and its two children is labeled by the graph edge deleted in the corresponding

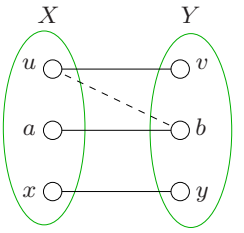


Fig. 2. Illustration of the proof of Lemma 1. The dashed line represents a deleted edge.

case. All solutions with size at most k are stored at the leaves of the search tree. Solution E'' corresponds to a path from the root to a leaf in the search tree. Assume that the edges in E'' are numbered according to their occurrences on this path, $e_1, e_2, \ldots, e_l$ with $l \leq k$, from the root to the leaf. Moreover, let e_i, e'_i with $1 \leq i \leq l$ be the edge pair of the induced $2K_2$ for which e_i has been deleted from B' and added to E''. Let $B_0 := B'$ and $B_i := B_{i-1} - e_i$ for $1 \leq i \leq l$. Obviously, $B_l = B''$. Moreover, let B'_i for $0 \leq i \leq l$ be the graph resulting by adding v to B_i and connecting v to $N_B(v)$. Clearly, $B'_0 = B$ and $B'_l = H$.

Since v is not in any $2K_2$ in B but in one in H there is a $2K_2$ containing v, we can assume that B'_j is the graph with the minimum index among $B'_1, \ldots, B'_l$ where v occurs in a $2K_2$ induced by edges $\{u, v\}$ and $\{a, b\}$ with $a, u \in X$ and $b, v \in Y$. Since v is not in any $2K_2$ in B'_{j-1} and E'' contains no edges incident to v, we have $\{a, v\} \notin E$ and $e_j = \{b, u\}$. According to the branching strategy stated above, $\{b, u\}$ has to form a $2K_2$ with an edge $\{x, y\}$ in B'_{j-1} with $x, y \notin \{a, v\}$. Assume that $x \in X$ and $y \in Y$. See Fig. 2 for an illustration of the subgraph of B'_i containing vertices u, v, x, y, a, b. Since v does not occur in any $2K_2$ in B, at least one of the edges $\{u, y\}$ and $\{v, x\}$ exists in B. If $\{v, x\} \in E$, then $\{b, x\} \in E$, since, otherwise, $\{v, x\}$ and $\{a, b\}$ would form a $2K_2$ containing v in B. Since v is not in any $2K_2$ in B'_{j-1}, the edge $\{b, x\}$ would exist in B'_{j-1}. This is a contradiction to the fact that $\{b, u\}$ and $\{x, y\}$ form a $2K_2$ in B_{j-1}. Thus, $\{v, x\} \notin E$. This implies that $\{u, y\} \in E$ and $\{u, y\}$ exists in B'_{j-1}. Again, we have a contradiction to the fact that $\{b, u\}$ and $\{x, y\}$ form a $2K_2$ in B_{j-1}. Therefore, H is a chain graph and E'' is a chain deletion set of B. □

Based on these two rules, we prove the size bound of the reduced graphs.

Theorem 1. *If a reduced instance (B, k) of* Chain Deletion *is a yes-instance, then B has at most $2k^2$ vertices. The kernelization runs in $O(mnk)$ time.*

Proof. Let $B = (X, Y, E)$ be a Chain Deletion instance that is reduced with respect to Rules 1 and 2 and has a chain deletion set E' with $|E'| \leq k$. We analyze in the following the size of X; the size bound of Y follows analogously. Since B is reduced with respect to Rule 2, every vertex from X is involved in at least one $2K_2$ in B. Moreover, due to Rule 1, there can be at most k^2 many $2K_2$'s in B with $|E'| \leq k$. Therefore, $|X| \leq k^2$.

We prove the running time by showing that Rules 1 and 2 can be exhaustively executed in $O(mnk)$ time: We first apply Rule 1 exhaustively and then apply

once Rule 2. For one application of Rule 1, iterate over all edges and, for each edge e, remove all neighbors of its endpoints. If there remain more than k edges, then e occurs in more than k $2K_2$'s and Rule 1 is applicable. Obviously, Rule 1 can be applied at most k times and needs $O(mnk)$ time.

To apply Rule 2, we compute all $2K_2$'s in the graph after exhaustive application of Rule 1. If a vertex v does not occur in $2K_2$, then apply Rule 2 to v. Thus, Rule 2 needs $O(mn)$ time. □

4 Split Deletion

A graph $G = (V, E)$ is called a *split graph* if there exists a partition (K, I) of V such that K is a clique and I is an independent set. Földes and Hammer [8] introduced this graph class in 1977 and gave the following forbidden subgraph characterization:

Lemma 2 ([8]). *A graph is a split graph if and only if it contains no induced $2K_2$, C_4, and C_5.*

Split Deletion is NP-complete [14]. We prove that Split Deletion admits a problem kernel with $O(k^4)$ vertices. Because split graphs are closed under the complement operation, the result holds for Split Completion as well.

We follow almost the same approach as in Sect. 3, namely, first get rid of the vertices that do not occur in any forbidden induced subgraph and then delete the edges which have to be in any size-$\leq k$ split deletion set. However, since split graphs have, besides $2K_2$, also C_4 and C_5 as forbidden subgraphs, we need additional data reduction rules and their correctness proofs are more complicated than in the case of Chain Deletion. In particular, how to deal with two edges that occur together in more than k forbidden subgraphs that, with the exception of these two edges, are pairwisely edge-disjoint, is a new task here. In general, given such two edges, we only know that at least one of them has to be deleted, but we cannot decide in polynomial time which one of them has to be deleted. Here, for $2K_2$-free graphs, we solve this problem by showing that such two edges can be replaced by a $2K_2$-similar gadget (see Rules 2 and 3).

We apply seven data reduction rules to a Split Deletion instance (G, k) and prove their correctness and running times, respectively. During the reduction process, whenever $k < 0$, we know that the given instance has no solution and report "No".

Rule 1. Delete the vertices from G that are not in any $2K_2$, C_4, and C_5.

Lemma 3. *Rule 1 is correct and one application of Rule 1 needs $O(mn^3)$ time.*

Proof. We prove this lemma by showing that the input graph $G = (V, E)$ has a size-$\leq k$ split deletion set iff the graph $G' = (V', E')$ after one application of Rule 1 has a size-$\leq k$ split deletion set.

"⇒": This direction is clearly correct since every induced subgraph of a split graph is a split graph.

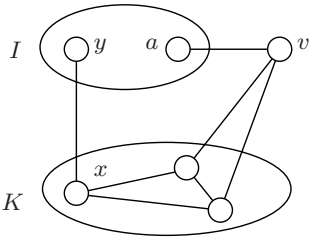


Fig. 3. Illustration of the proof of Lemma 3

"⇐": Suppose that G' has a split deletion set S with $|S| \leq k$ and let $G'' = (V'', E'')$ denote the graph after deleting the edges in S from G', where K and I are the clique and the independent set of G''. If the graph resulting by adding v to G'' and connecting v to $N_G(v)$ remains a split graph, then we are done; otherwise, there exist some vertex $x \in K$ with $\{x, v\} \notin E$ and some vertex $a \in I$ with $\{a, v\} \in E$. See Fig. 3 for an illustration. W.l.o.g., we can assume that, in G'', the set $K \cap N_G(v)$ is maximal in the sense that, for each vertex u from $I \cap N_G(v)$, there is a vertex in K that is not adjacent to u. Moreover, we can also assume that every vertex in K has a neighbor in $I \cup \{v\}$ and let y be a neighbor of x in I. Next, we prove the following claim.

Claim 1. The set $N_G(v)$ is a clique in G.

Proof of Claim 1: We prove the claim by contradiction. Suppose that $a_1, a_2 \in N_G(v)$ with $\{a_1, a_2\} \notin E$. Observe that $|K \setminus N_G(v)| \leq 1$. To see this, suppose that there exists, besides x, another vertex x' in $K \setminus N_G(v)$. Since v does not occur in any $2K_2$ in G, at least two of the edges $\{a_1, x\}$, $\{a_1, x'\}$, $\{a_2, x\}$, and $\{a_2, x'\}$ have to exist in G. These edges and the edges $\{x, x'\}$, $\{a_1, v\}$, and $\{a_2, v\}$ induce at least one C_4 or C_5 containing v, a contradiction to the fact that v does not occur in any C_4 and C_5 in G. Thus, $|K \setminus N_G(v)| \leq 1$.

Next, we show that $y \notin N_G(v)$. Suppose that $y \in N_G(v)$. Then there exists a vertex $z \in K$ with $\{y, z\} \notin E$; otherwise, $K \cap N_G(v)$ would not be maximal. However, due to the fact $|K \setminus N_G(v)| \leq 1$, we have $z \in N_G(v)$ and, thus, a C_4 with v, z, x, y in G, a contradiction to the fact that v does not occur in any C_4. Since $x, y \notin N_G(v)$ and v is not contained in any $2K_2$ in G, at least two of the edges $\{x, a_1\}$, $\{x, a_2\}$, $\{y, a_1\}$, and $\{y, a_2\}$ have to exist in E and, thus, we have a C_4 or C_5 containing v in G, a contradiction to the fact that v does not occur in any C_4 and C_5. This concludes the proof of Claim 1.

Now we show that, in the case that $N_G(v)$ is a clique, we can construct a split deletion set S' from S for G with $|S'| \leq |S| \leq k$. Consider the split graph $H = (V_H, E_H)$ where the clique K' of H consists of the vertices in $N_G(v) \cup X$ with $X := \{v \in K \mid v \in (\bigcap_{u \in N_G(v)} N_G(u))\}$ and the independent set I' of H consists of the vertices in $\{v\} \cup (I \setminus N_G(v)) \cup (K \setminus X)$. The set E_H contains all possible edges between the vertices in K' and all edges in E between K' and I'. Since $N_G(v)$ is a clique, there exists a split deletion set S' transforming G into H.

To show $|S'| \leq |S|$, it suffices to show $|E^1| + |E^2| \geq |E^3| + |E^4|$, where E^1 is the set of the edges in E between the vertices in $N'(v) := N_G(v) \cap I$, E^2 the set of the edges in E between $N'(v)$ and $I \setminus N_G(v)$, E^3 the set of the edges in E between $K \setminus (X \cup N_G(v))$ and $I \setminus N'(v)$, and E^4 the set of the edges in E between the vertices in $K \setminus (X \cup N_G(v))$.

First we claim that $|N'(v)| \geq |K \setminus (X \cup N_G(v))|$. Since both sets are cliques in G, this claim implies $|E^1| \geq |E^4|$. Suppose that the claim is not true. Then, by the pigeonhole principle, there exist at least two distinct vertices $x, y \in (K \setminus (X \cup N_G(v)))$ with $\{x, a\} \notin E$ and $\{y, a\} \notin E$ for a vertex $a \in N'(v)$. This implies that we have a $2K_2$ with edges $\{x, y\}$ and $\{a, v\}$ in G, a contradiction to the fact that v does not occur in a $2K_2$ in G.

Next we show that $|E^2| \geq |E^3|$. Consider a vertex $x \in (K \setminus (X \cup N_G(v)))$ with a neighbor y in I. From the definition of X, we have $N'(v) \setminus N_G(x) \neq \emptyset$ and let $a \in (N'(v) \setminus N_G(x))$. We have $\{a, y\} \in E$, since, otherwise, $\{x, y\}$ and $\{a, v\}$ would induce a $2K_2$ in G. Thus, for each edge from E^3 that is incident to x, there exists at least one edge from E^2 incident to a' for every vertex $a' \in (N'(v) \setminus N_G(x))$. Moreover, for each two distinct vertices $w, w' \in (K \setminus (X \cup N_G(v)))$, the sets $N'(v) \setminus N_G(w)$ and $N'(v) \setminus N_G(w')$ are disjoint, since, otherwise, there would be a $2K_2$ involving v, w, w' in G. Therefore, we can conclude that $|E^2| \geq |E^3|$ and $|S'| \leq |S|$.

The running time of Rule 1 follows from the observation that all $2K_2$'s, C_4's, and C_5's of G can be enumerated in $O(mn^3)$ time. □

> **Rule 2.** If two edges $\{u, v\}$ and $\{u, w\}$ occur together in more than k C_4's, then delete $\{u, v\}$ and $\{u, w\}$ from G and add two edges $\{a, v\}$ and $\{b, w\}$ to G with a, b being two new degree-one vertices.

Lemma 4. *Rule 2 is correct and one application of Rule 2 needs $O(m^2 n)$ time.*

Proof. Let $\{u, v\}$ and $\{u, w\}$ be the two edges to which Rule 2 has been applied. We use $X := \{x_1, x_2, \ldots, x_{k+1}\}$ to denote the set of the fourth vertices of $k + 1$ arbitrarily chosen C_4's containing both $\{u, v\}$ and $\{u, w\}$. Let $G' = (V', E')$ denote the resulting graph after applying Rule 2 to $\{u, v\}$ and $\{u, w\}$. We prove the correctness of Rule 2 by showing that we can transform a split deletion set S of the original graph G into a split deletion set S' of G' with $|S'| = |S| \leq k$ and vice versa. We show here only the direction from S to S'; the reverse direction can be proven in a similar way.

We use K and I to denote the clique and the independent set of the resulting split graph after deleting S from G. Clearly, at least one of v and w has to be in I. Then, at least one of $x_1, \ldots, x_{k+1}$ has to be in K and $u \in I$. If $v \in I$ and $w \in K$, then $S' := S \setminus \{\{u, v\}\} \cup \{\{a, v\}\}$ is a split deletion set for G'. To see this, observe that, due to the existence of $x_1, \ldots, x_{k+1}$, both a and b have to be in the independent set of any split graph resulting by deleting at most k edges from G'. This argument works also for the case $v, w \in I$ with the only difference lying in the construction of S', namely, $S' := S \setminus \{\{u, v\}, \{u, w\}\} \cup \{\{a, v\}, \{b, w\}\}$.

To apply Rule 2, we iterate over all two adjacent edges $\{u, v\}$ and $\{u, w\}$ with $\{v, w\} \notin E$. For each such edge pair, delete the vertices in $N_G[u] \setminus \{v, w\}$

and count the common neighbors of v and w in the resulting graph. If the number of common neighbors exceeds k, then Rule 2 is applicable to these two edges. □

Rule 3. If two adjacent edges e and e' occur together in more than k C_5's that, with the exception of e and e', are pairwisely edge-disjoint, then delete e and e', add e and e' to the split deletion set, and decrease the parameter k by two.

Lemma 5. *Rule 3 is correct and one application Of Rule 3 needs* $O(m^3\sqrt{n})$ *time.*

Proof. Suppose that edges $e = \{u, v\}$ and $e' = \{u, w\}$ are two edges that satisfy the precondition of Rule 3. Let $\{x_1, y_1\}, \{x_2, y_2\}, \ldots, \{x_{k+1}, y_{k+1}\}$ be the edges which are from $k + 1$ arbitrary C_5's containing e and e' and whose endpoints are not from $\{v, w\}$. Clearly, at least one of $x_1, \ldots, x_{k+1}$ and at least one of $y_1, \ldots, y_{k+1}$ have to be in the clique of any split graph resulting by deleting at most k edges from G. Then, both v, w have to be in the independent set that contains u as well. Hence, we have to delete e and e'.

To apply this rule, we iterate over all two adjacent edges whose endpoints do not induce a triangle. For each pair $e = \{u, v\}$ and $e' = \{u, w\}$, we construct a subgraph of G having only the vertices in $N(v) \setminus (N[u] \cup N(w))$ and $N(w) \setminus (N[u] \cup N(v))$ and the edges between these two sets. If this bipartite subgraph has a matching of size more than k, then this rule is applicable to e and e'. The running time follows from the running time $O(m\sqrt{n})$ for computing maximum bipartite matchings [4]. □

Rule 4. If an edge e occurs in more than k $2K_2$'s, then delete e from G, add e to the split deletion set, and decrease the parameter k by one.

Rule 5. If an edge e occurs in more than k C_4's that, with the only exception of e, are pairwisely edge-disjoint, then delete e from G, add e to the split deletion set, and decrease the parameter k by one.

Rule 6. If an edge e occurs in more than k C_5's that, with the only exception of e, are pairwisely edge-disjoint, then delete e from G, add e to the split deletion set, and decrease the parameter k by one.

Lemma 6. *Rules 4, 5, and 6 are correct and one application of these rules needs* $O(mn)$, $O(m^2\sqrt{n})$, *and* $O(mn^3)$ *time, respectively.*

Proof. Clearly, these rules are correct. We give here only a proof of the running time $O(m^2\sqrt{n})$ of Rule 5. To apply Rule 5, we iterate over all graph edges and, for each edge $e = \{u, v\}$, keep only the vertices in $N(u)\setminus N[v]$ and $N(v)\setminus N[u]$ and the edges between these two sets. Finally, we compute a maximum matching of the remaining bipartite graph. If the matching has a size more than k, then Rule 5 can be applied to e. The running time follows from the running time $O(m\sqrt{n})$ for computing maximum bipartite matchings [4]. □

Rule 7. If three edges $\{u, v\}$, $\{v, w\}$, and $\{w, x\}$ occur together in more than k C_5's, then delete $\{v, w\}$ from G, add $\{v, w\}$ to the split deletion set, and decrease the parameter k by one.

The following lemma can be shown with the same arguments as used in the proofs of Lemmas 4 and 6.

Lemma 7. *Rule 7 is correct and one application of Rule 7 needs $O(m^3n)$ time.*

Now, we arrive at the central result of this section.

Theorem 2. *If a reduced instance $(G = (V, E), k)$ of* Split Deletion *is a yes-instance, then $|V| = O(k^4)$. The kernelization runs in $O(m^4\sqrt{n})$ time.*

Proof. Suppose that a reduced graph G has a split deletion set S with $|S| \leq k$. Due to Rule 1, every vertex v in G has to be in a $2K_2$, C_4, or C_5. In the following we give an upper bound on the number of the vertices which are contained in C_5's. Clearly, every C_5 in G has to contain at least one edge from S. Due to Rule 6 every edge e of S can be in at most k C_5's that have only e in common. Let A denote the set of these C_5's. There are at most $4k + 1$ edges in the C_5's from A. Since G is reduced with respect to Rule 3, each edge e' from the C_5's in A with $e' \neq e$ can form together with e at most k C_5's that, with the exception of e and e', are pairwisely edge-disjoint. Add these C_5's with both e and e' to A. Now A contains at most $4k^2$ many C_5's. These C_5's contain at most $12k^2$ many induced length-two paths that contain e. Due to Rule 7, each of these P_3's can be contained in at most k C_5's. Therefore, edge e can be contained in at most $12k^3$ many C_5's. For $|S| \leq k$, we have at most $12k^4$ many C_5's in G which gives an upper bound of $36k^4 + 2k$ on the number of the vertices contained in C_5's: $|V \setminus V(S)| \leq 36k^4$ and $|V(S)| \leq 2k$ with $V(S)$ being the vertex set of S. With similar arguments, the number of the vertices in C_4's and $2K_2$'s can also be upper-bounded by $O(k^3)$ and $O(k^2)$, respectively. This gives the size bound claimed in the theorem. The running time follows from Lemmas 3 to 7 and the fact that the seven rules can altogether be executed at most m times. □

5 Threshold Deletion and Co-trivially Perfect Deletion

A graph G is a *threshold graph* iff G contains no induced $2K_2$, C_4, and P_4, while a *co-trivially perfect graph* contains no induced $2K_2$ and P_4 [1]. Threshold graphs have been extensively studied in literature [1]. See the monograph of Mahadev and Peled [12] for more results on threshold graphs. Margot [13] showed that Threshold Deletion is NP-complete. Yannakakis [19] showed that Co-Trivially Perfect Deletion is NP-complete. Clearly, threshold graphs form a subclass of both split graphs and co-trivially perfect graphs. We prove that both problems admit polynomial-time kernelizations with $O(k^3)$-vertex problem kernels. These results hold for Threshold Completion and Trivially Perfect Completion as well, since threshold graphs are closed under complementation and the complement graph of a co-trivially perfect graph is a trivially perfect graph.

The basic idea behind the kernelizations is almost the same as for Split Deletion. Therefore we present only the data reduction rules for Threshold Deletion. The explicit description of the data reduction rules for Co-Trivially Perfect Deletion is deferred to the full version of this paper.

Rule 1. Delete the vertices from G that are not in any $2K_2$, C_4, and P_4.
Rule 2. If an edge e occurs in more than k $2K_2$'s, then delete e from G, add it the threshold deletion set, and decrease the parameter k by one.
Rule 3. If an edge e occurs in more than k C_4's that, with the only exception of e, are pairwisely edge-disjoint, then the given instance has no solution and report "No".
Rule 4. If two edges e and e' occur together in more than k C_4's, then delete e and e' from G, add both to the threshold deletion set, and decrease the parameter k by two.
Rule 5. If an edge e is the middle edge of more than k P_4's that, with the only exception of e, are pairwisely edge-disjoint, then the given instance has no solution and report "No".
Rule 6. If an edge e occurs in more than k P_4's as a side edge, then delete e, add e to the threshold deletion set, and decrease the parameter k by one.
Rule 7. If two edges e and e' occur together in more than k P_4's where e' is the middle edge, then remove e from G, add e to the threshold deletion set, and decrease the parameter k by one.

Theorem 3. *If a reduced instance* $(G = (V, E), k)$ *of* Threshold Deletion *is a yes-instance, then* $|V| = O(k^3)$. *The kernelization runs in* $O(kn^4)$ *time.*

Based on five data reduction rules that are very similar to Rules 1, 2, 5, 6, and 7 described above, we can also achieve a kernelization for Co-Trivially Perfect Deletion.

Theorem 4. *If a reduced instance* $(G = (V, E), k)$ *of* Co-Trivially Perfect Deletion *is a yes-instance, then* $|V| = O(k^3)$. *The kernelization runs in* $O(kn^4)$ *time.*

6 Outlook

In this work, we studied several edge deletion problems for generating $2K_2$-free graphs and obtained polynomial-time kernelization algorithms for them. An interesting open problem is whether the approach adopted here, namely, first to get rid of the vertices that are not in any forbidden subgraph and then to delete the edges which are contained in too many forbidden subgraphs, also applies to edge deletion problems for generating $2K_2$-free graphs, even when the set of forbidden subgraphs is infinite. Moreover, another challenging task would be to improve the kernel sizes achieved here to linear size (if possible) and the running times of the kernelizations, as done for Cluster Editing [6,10,17].

References

1. Brandstädt, A., Le, V.B., Spinrad, J.P.: Graph Classes: a Survey. In: SIAM Monographs on Discrete Mathematics and Applications (1999)
2. Burzyn, P., Bonomo, F., Durán, G.: NP-completeness results for edge modification problems. Discrete Applied Mathematics 154, 1824–1844 (2006)

3. Cai, L.: Fixed-parameter tractability of graph modification problems for hereditary properties. Information Processcing Letters 58, 171–176 (1996)
4. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms. MIT Press and McGraw-Hill (2001)
5. Downey, R.G., Fellows, M.R.: Parameterized Complexity. Springer, Heidelberg (1999)
6. Fellows, M.R., Langston, M.A., Rosamond, F., Shaw, P.: Polynomial-time linear kernelization for Cluster Editing. In: Csuhaj-Varjú, E., Ésik, Z. (eds.) FCT 2007. LNCS, vol. 4639, Springer, Heidelberg (2007)
7. Flum, J., Grohe, M.: Parameterized Complexity Theory. Springer, Heidelberg (2006)
8. Földes, S., Hammer, P.L.: Split graphs. Congressus Numerantium 19, 311–315 (1977)
9. Gramm, J., Guo, J., Hüffner, F., Niedermeier, R.: Graph-modeled data clustering: Exact algorithms for clique generation. Theory of Computing Systems 38(4), 373–392 (2005)
10. Guo, J.: A more effective linear kernelization for Cluster Editing. In: Chen, B., Paterson, M., Zhang, G. (eds.) ESCAPE 2007. LNCS, vol. 4614, pp. 36–47. Springer, Heidelberg (2007)
11. Guo, J., Niedermeier, R.: Invitation to data reduction and problem kernelization. ACM SIGACT News 38(1), 31–45 (2007)
12. Mahadev, N.V.R., Peled, U.N.: Threshold Graphs und Related Topics. Annals of Discrete Mathematics 56 (1995)
13. Margot, F.: Some complexity results about threshold graphs. Discrete Applied Mathematics 49(1-3), 299–308 (1994)
14. Natanzon, A., Shamir, R., Sharan, R.: Complexity classification of some edge modification problems. Discrete Applied Mathematics 113, 109–128 (2001)
15. Niedermeier, R.: Invitation to Fixed-Parameter Algorithms. Oxford University Press, Oxford (2006)
16. Niedermeier, R., Rossmanith, P.: A general method to speed up fixed-parameter-tractable algorithms. Inf. Process. Lett. 73, 125–129 (2000)
17. Protti, F., da Silva, M.D., Szwarcfiter, J.L.: Applying modular decomposition to parameterized bicluster editing. In: Bodlaender, H.L., Langston, M.A. (eds.) IWPEC 2006. LNCS, vol. 4169, pp. 1–12. Springer, Heidelberg (2006)
18. Sharan, R.: Graph Modification Problems and Their Applications to Genomic Research. PhD thesis, School of Computer Science, Tel-Aviv University (2002)
19. Yannakakis, M.: Computing the minimum fill-in is NP-complet. SIAM Journal on Algebraic and Discrete Methods 2(1), 297–309 (1981)
20. Yannakakis, M.: Edge-deletion problems. SIAM Journal on Computing 10(2), 297–309 (1981)

Author Index