Series in
Intelligent Control and Intelligent Automation
Vol. 13

Network-Based Distributed Planning Using Coevolutionary Algorithms

Raj Subbu
Arthur C Sanderson

Network-Based Distributed Planning Using Coevolutionary Algorithms

SERIES IN INTELLIGENT CONTROL AND INTELLIGENT AUTOMATION

Editor-in-Charge: Fei-Yue Wang
(*University of Arizona*)

Vol. 1: Reliable Plan Selection by Intelligent Machines
(*J E McInroy, J C Musto, and G N Saridis*)

Vol. 2: Design of Intelligent Control Systems Based on Hierachical Stochastic Automata (*P Lima and G N Saridis*)

Vol. 3: Intelligent Task Planning Using Fuzzy Petri Nets
(*T Cao and A C Sanderson*)

Vol. 4: Advanced Studies in Flexible Robotic Manipulators: Modeling, Design, Control and Applications (*F Y Wang*)

Vol. 6: Modeling, Simulation, and Control of Flexible Manufacturing Systems: A Petri Net Approach (*M Zhou and K Venkatesh*)

Vol. 7: Intelligent Control: Principles, Techniques, and Applications (*Z-X Cai*)

Vol. 10: Autonomous Rock Excavation: Intelligent Control Techniques and Experimentation (*X Shi, P J A Lever and F Y Wang*)

Vol. 11: Multisensor Fusion: A Minimal Representation Framework
(*R Joshi and A C Sanderson*)

Vol. 12: Entropy in Control Engineering
(*George N Saridis*)

Forthcoming volumes:

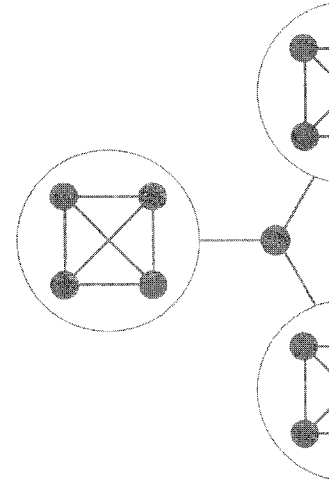
Vol. 5: Computational Foundations for Intelligent Systems (*S J Yakowitz*)

Vol. 8: Advanced Topics in Computer Vision and Pattern Recognition
(*E Sung, D Mital, E K Teoh, H Wang, and Z Li*)

Vol. 9: Petri Nets for Supervisory Control of Discrete Event Systems: A Structural Approach (*A Giua and F DiCesare*)

Series in
Intelligent Control and Intelligent Automation
Vol. 13

Network-Based Distributed Planning Using Coevolutionary Algorithms



Raj Subbu
General Electric Global Research, USA

Arthur C Sanderson
Rensselaer Polytechnic Institute, USA

World Scientific
NEW JERSEY • LONDON • SINGAPORE • SHANGHAI • HONG KONG • TAIPEI • BANGALORE

Published by

World Scientific Publishing Co. Pte. Ltd.

5 Toh Tuck Link, Singapore 596224

USA office: Suite 202, 1060 Main Street, River Edge, NJ 07661

UK office: 57 Shelton Street, Covent Garden, London WC2H 9HE

British Library Cataloguing-in-Publication Data
A catalogue record for this book is available from the British Library.

NETWORK-BASED DISTRIBUTED PLANNING USING COEVOLUTIONARY ALGORITHMS
Series in Intelligent Control and Intelligent Automation — Vol. 13

ISBN 981-238-754-4

Printed in Singapore by World Scientific Printers (S) Pte Ltd

To Roopa and my parents
for their loving support of all my projects
- Raj Subbu

This page is intentionally left blank

Foreword

Modern enterprises and organizations interested in participating in today's global business environment are most likely to employ some form of distributed computing to assist in accomplishing their goals. Perhaps labels such as e-commerce, B2B-commerce (business to business), C2B-commerce (consumer to business), e-engineering, etc. are applied to the context, but underlying the context is a core of advanced technologies and within that core is likely to be a form of distributed computing.

Consider a consumer searching for a product over the web by placing requests for information through a browser to a search engine. These terms that we now use and this form of product search were nearly unheard of as recently as a decade ago, but are relatively commonplace today and will be even more so in the next five years.

Consider a large auto company whose suppliers are coordinated into a common market environment where component and subsystem pricing as well as availability are readily visible to the auto company's material procurement staff.

Both of these examples rely on forms of distributed computing. Each example supports a manual/user-oriented search over vast amounts of information available through linked computers and their databases. The genesis of their respective underlying technologies goes back to perhaps a decade ago and we see such systems in use today.

Now consider the future!

Suppose that we could automate that search process rather than being in front of our computer terminal directing it at each step of its execution. In the former case, we would configure a search with the product characteristics (e.g. size, weight, speed, brand name) desired and then let the computer and computing system execute our search. In the latter case, a

similar search process might apply.

The future is here!

The technologies needed to accomplish large-scale automated searches have been developed and proved and are being migrated to commercial applications. One part of this suite of technologies is known as intelligent computational agents. These can be sent over a network environment to access and evaluate information in distributed databases for the "fit" to our constraints.

Suppose that we seek yet more in our search process. Suppose that we would like to tell our computer system what we need and have it come back with the results, narrowed to our specific goals. For example, the search for the product characteristics needed (i.e. size, speed) may return many results that "fit." If we also direct the search process to find those products that fit but are, as well, the "least cost" or "readily available" we can expect fewer results back from our system's search, but the results should be very close to our desires. This means that the computer system has accessed distributed databases, developed information of the type we seek, evaluated perhaps millions of results and continued to search until it found the best result or until some stopping procedure told it that we had something close enough to our needs. Such additional search requirements for large-scale decision problems often encounter non-linearities in meeting our objectives where good results must be sought over multiple domains and an ideal result in one search domain may not be so attractive when considered in another domain. If we were to do all of this manually, we would not likely be so thorough, not so diligent in our efforts, nor as patient as the computer system.

It is therefore very important that this search and evaluation process be set up to be thorough and efficient. The large-scale nature of the possible search space over the network and the complexities of a possible industrial planning problem being addressed can tax the capabilities of even fast computers. So, we would not want to overlook possible good solutions (i.e. we want to be thorough in the search), but also would like results back in a reasonable period of time (i.e. we must be efficient). Many algorithms or optimization search approaches would be considered "thorough" when applied to a properly structured problem. One especially difficult aspect of the need to be "efficient," however, is the delay inherent in computer networks due to the number of users, the sizes of user information demands, downtime, etc. Consequently, a number of researchers undertook the task of finding efficient algorithms for optimum seeking searches of distributed

information sources on the web.

Coevolutionary algorithms, coded and imbedded in the computer system, are an excellent means for automated coordination and re-direction of the search based upon the intermediate results as required in the above described process. These efficiencies are achieved by situating the search components closer to the sources of information in delay-prone networks. Such algorithms are therefore distinguished from evolutionary algorithms in that the algorithm's computational work is distributed to several sites in a network of computers in order to improve speed and efficiency of its execution with large numbers of users and databases; hence a scalable version for commercial applications. The result of integrating these technologies is the automated and intelligent search process that can both improve the efficiency of decision and planning problems as well as the solution(s) itself when we face network-based planning and decision environments.

This book's authors address these technologies focused on decentralized search and planning systems using evolutionary algorithms. They also explore their use in large-scale commercial applications for planning problems. Drs. Subbu and Sanderson are members of a select set of people worldwide who are doing active research work in Coevolutionary Multi-agent Systems and in Network-based Distributed Decision-Making Systems. This is a particularly exciting and challenging research area due to the current and future emphasis on network-based information systems to support various businesses and supply chain applications with tremendous potential benefits in competitiveness, effectiveness, and productivity. Their work is at a crucial core area of this field where such information technology systems must be designed and implemented in a manner allowing for large numbers of users, i.e. scalable, and also be reliable and robust in a potentially unpredictable network environment. Their initial work developed within the Electronics Agile Manufacturing Research Institute at Rensselaer Polytechnic Institute. Under a National Science Foundation (NSF) contract (DMI - 9320955) and subsequent contracts on Scalable Enterprises (DMI - 0075524, DMI - 0121902), the EAMRI researchers, led by Drs. Subbu and Sanderson developed the centralized optimization formulation and solution approach and extended it to co-evolutionary domains. It was applied to the electronics concurrent engineering problem described in the book, and is currently patented with a second patent pending. Their publications include seminal papers on the theory and practice of evolutionary and co-evolutionary algorithms that are at the core of the patented technology and their further research. In addition to the NSF support for their work, the

EAMRI is supported with resources provided by collaborating companies including Lucent Technologies, Cisco Systems, Pitney Bowes, Benchmark Electronics, Vermont Circuits, Hughes, and Texas Instruments.

This book, for the technical reader, is an exciting document about exciting technologies in this network distributed computing domain.

Dr. Robert J. Graves

Director,
Rensselaer's Electronics Agile Manufacturing Research Institute
Professor Emeritus, Rensselaer Polytechnic Institute
and
Krehbiel Professor of Emerging Technologies
Thayer School of Engineering
Dartmouth College

Preface

The efficient development and implementation of network-based planning applications is increasingly dependent on the availability of generalized tools that support scalable and efficient network-distributed search and decision-making. In one approach to network-based planning, an objective function guides the distributed decision-making process, and the computation of the objective function requires efficient retrieval and interpretation of information from logically interrelated databases distributed over an inhomegenous network environment. The primary focus of this book is to describe fundamental principles of an approach to network-based search and planning based on evolutionary algorithms.

In this approach, efficient and scalable algorithms for distributed, network-based decision-making based on objective functions are developed in a network-distributed environment where internode communications are a primary factor in system performance. The decision-making algorithms that function in this environment must demonstrate superior performance while simultaneously economizing internode communications.

A class of distributed decision problems is developed based on the case of integrated design, supplier, and manufacturing planning for modular products, where suppliers and manufacturing resources are distributed. A formal model for this class of distributed decision problems is developed as a set of coupled nonlinear assignment problems. This model, called the *design-supplier-manufacturing planning decision problem*, is an example of a complex discrete optimization problem. The nonlinearities and coupling inherent in the problem class complicate application of exact optimization algorithms. Evolutionary algorithms however, are highly adaptable, place minimal restrictions on problem structure, and provide efficient and reliable solutions. A novel *coevolutionary algorithm*, based on distributed evolution-

ary algorithm components and software agents, is proposed as a network-efficient strategy for concurrent, cooperative exploration of a highly coupled space of design, supplier, and manufacturing decisions.

A theoretical foundation for this class of coevolutionary algorithms is developed using techniques from stochastic process theory and mathematical analysis. In this framework, the distributed computation is described in terms of construction and evolution of sampling distributions over the feasible space. Convergence and convergence rate analyses are pursued for certain basic classes of objective functions, and analytical and simulation techniques are used to evaluate the network-based performance of the algorithms in this class.

As a case study in distributed, network-based decision-making, an implementation and detailed evaluation of the coevolutionary decision-making framework suited to distributed network-enabled design and manufacturing organizations is presented. This implementation, the Coevolutionary Virtual Design Environment (CVDE), utilizes distributed evolutionary agents and mobile agents as principal entities that generate and execute queries and transactions among distributed applications and databases to support a global optimization of design, supplier, and manufacturing planning decisions. In this framework, an electronic interchange of design, supplier, and manufacturing information facilitates a concurrent, cooperative, network-efficient evolutionary search for promising planning alternatives.

The methodology presented in this book can have a fundamental impact on the principles and practice of engineering in industrial product development in the network-based distributed environment that is emerging within and among corporate enterprise systems. In addition, the conceptual framework of the approach to distributed decision systems presented in this book may have much wider implications for network-based systems ranging from intelligent agent-based browser systems, to enhanced consumer and business services, and intelligent search techniques in scientific and commercial databases.

Acknowledgments

The authors would like to thank World Scientific publishers, series editor Feiyue Wang, and editor Steven Patt for selecting this work as part of the series on intelligent control and intelligent automation.

This book is based on the doctoral research by the first author during the years 1996–2000 under the guidance of the second author, while at the Electronics Agile Manufacturing Research Institute (EAMRI) in Rensselaer

Polytechnic Institute.

The authors would like to acknowledge sources of support for this work and for the development of the manuscript. Research grants DMI - 9320955 and DMI - 0075524 from the National Science Foundation (NSF) gave valuable multi-year support to the first author's doctoral research. The follow-on research grant DMI - 0121902 from the NSF towards the collaborative project in Scalable Enterprise Systems between Rensselaer Polytechnic Institute and General Electric Research has in part supported the development of this manuscript. The authors thank Robert J. Graves and Alan A. Desrochers for actively championing and supporting the goals of this research, and for serving on the doctoral thesis committee. The authors also thank Piero P. Bonissone and Badrinath Roysam for serving on the doctoral thesis committee, and for their valuable suggestions during the development of this work. The authors acknowledge the helpful support of the departments of Electrical, Computer and Systems Engineering, and Decision Sciences and Engineering Systems at Rensselaer Polytechnic Institute.

The first author appreciates the helpful cooperation of his colleagues and managers at General Electric Research for fostering an intellectually stimulating, collaborative, and supportive environment. Finally on a personal note, the first author would like to thank his wife Roopa for her loving support and his parents for their unfailing inspiration, which have made so many things possible.

This page is intentionally left blank

Contents

Chapter 1

Introduction

1.1 Motivation

Advances in information technologies are driving fundamental changes in the processes and organizations of global enterprises. In an increasingly networked global marketplace, products and services are seldom created in isolation and are instead being realized through strategic and dynamic partnerships between suppliers, contract manufacturers, and customers. However, as the numbers of these distinct entities increase and they get more distributed, the complexity of forming efficient partnerships grows; it becomes more difficult to make ideal assignments (partnerships) with respect to multiple criteria including cost and time. Fundamental to this complexity is that each assignment has the potential to affect overall product cost, and product realization time, and therefore assignments cannot be considered independent of one another. Due to this complexity it is increasingly impossible to make these dynamic partnerships purely on the basis of prior experience, and it becomes necessary to develop efficient planning systems that can automate significant portions of the overall decision task. Such systems access information available at multiple, logically interrelated, distributed databases in order to evaluate the consequences of the assignments.

Decision automation is achieved through efficient search of the distributed space of planning decisions, guided by an objective function whose computation requires information efficiently gleaned from databases distributed over an inhomogeneous network environment. Generalized tools that support scalable and efficient network-distributed search and decision-making are therefore critical.

Efficient planning and decision-making in an inhomogeneous network environment requires transcending technical hurdles at multiple hierarchi-

cal levels. At the *network level* a principal issue is the integration of heterogeneous local-area networks so data can be exchanged seamlessly and efficiently across organizational boundaries. At a higher level, the *middleware level*, a principal issue is the design of software systems and protocols that facilitate efficient communication between distributed databases and computing applications. A principal issue at the next level up, decision-making in a heterogeneous network-distributed environment, is of particular interest in this book. The challenge at this level is to develop computationally efficient optimization and decision-making algorithms that access logically related information from distributed sites and manipulate it for some decision-making purpose. An efficient decision-making scheme functioning in this network-distributed environment leverages resources offered at the middle-ware level, and the middle-ware level in turn leverages resources available at the network level.

In a network-distributed environment internode communications are a primary factor in system performance, and the decision-making algorithms that function in this environment must demonstrate superior performance while simultaneously economizing internode communications. These algorithms must also scale well with the number of distributed information sites that need to be accessed for decision-making. Development of the architecture and algorithms that support efficient distributed decision-making at an enterprise level, is the focus of this book.

1.2 Approach

The goal is a systematic development of scalable and efficient algorithms that support distributed, network-based decision-making. First, a class of distributed decision problems is selected. This class of decision problems is the integrated design, supplier, and manufacturing planning for modular products, where suppliers and manufacturing resources are distributed (see Figure 1.1).

The mathematical structure of this planning task is given by $\min\{\psi(x) : \mathbf{g}(x) = 0, x \in \mathbb{Z}_+^n\}$, where x represents a complete decision vector, $\psi(\cdot)$ is a nonlinear objective function, and $\mathbf{g}(\cdot)$ is an m-vector of constraint functions. A complete decision vector x corresponds to a structured selection from an available set of distributed resources. A particular decision problem in this class is of interest throughout this book and consists of three assignment problems (A_1, A_2, A_3). The assignment problem A_1 is the assignment

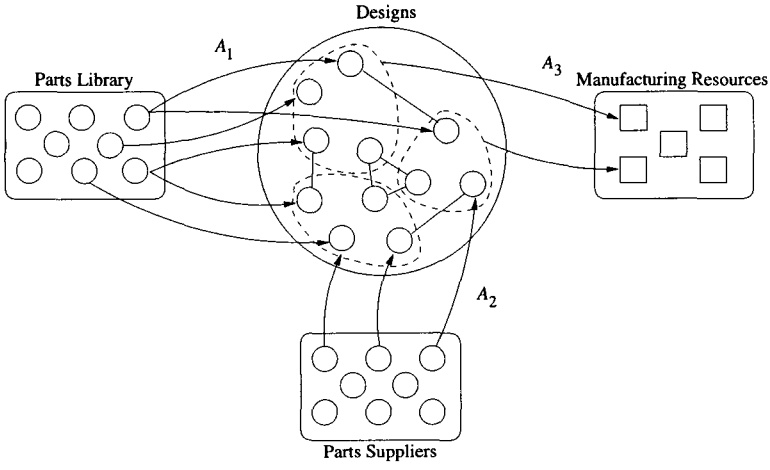


Fig. 1.1 Structure of the integrated design, supplier, and manufacturing planning decision problem. Lines with arrow heads indicate assignments. Dashed lines indicate aggregates. Identical parts in various designs have solid lines between them.

of parts (from a parts library) to a design that satisfies a predetermined functional specification. Multiple designs that satisfy the functional specification are possible. The assignment problem A_2 is the assignment of suppliers (from a list of available suppliers) who will supply the parts in a design, and the assignment problem A_3 is the assignment of designs to available manufacturing resources. Each of these assignments contributes to overall product cost and product realization time, and cannot be considered independent of one another. Therefore, the assignment problem triple (A_1, A_2, A_3) constitutes a set of highly coupled problems. Also, the assignments have nonlinear (cannot be evaluated as weighted sums) effects on product cost, and product realization time. A formal model of this coupled nonlinear assignment problem class is developed and called the *design-supplier-manufacturing planning decision problem*, and the formulation reveals a complex structure with multiple variables, constraints, and performance criteria.

An assignment problem is a discrete optimization problem. Discrete optimization problems incur a heavy penalty due to dimensionality, since problem sizes grow exponentially with the number of options along each dimension. Eventually, as the problem size grows or due to nonlinearities, it becomes infeasible to utilize exact optimization algorithms and one is forced to consider approximate algorithms. The *design-supplier-manufacturing*

planning decision problem consists of three assignment problems that are highly coupled and nonlinear, and is an example of a hard discrete optimization problem. The highly coupled and nonlinear characteristics inherent in this problem class complicate application of exact optimization algorithms.

Evolutionary algorithms are capable of generating reliable solutions in a time-efficient manner even for very large and complex discrete problems. Moreover, they are highly adaptable and place minimal restrictions on the nature of an optimization problem. The *design-supplier-manufacturing planning decision problem*, an example of a complex discrete optimization problem, is a good candidate for evolutionary optimization.

An integrated design, supplier, and manufacturing planner functions in a network-distributed environment with distributed databases and applications that are accessed numerous times for decision-making. In this network environment internode communications are a primary factor in system performance, and the algorithms that function in this environment must exploit data locality to improve computational efficiency. Importantly, these algorithms must demonstrate superior performance while simultaneously economizing internode communications. This motivates development of a novel class of *coevolutionary algorithms* based on distributed evolutionary algorithm components and software agents as a network-efficient strategy for concurrent, cooperative exploration of a highly coupled space of design, supplier, and manufacturing decisions. In this computational model, the search variables are *partitioned* among p nodes (see Figure 1.2). An evolutionary algorithm at each of the p nodes performs a local evolutionary search based on its own set of primary variables using local and rapidly accessible information (from a local database) while the secondary variable set at each node is clamped during this phase. An intercommunication between the nodes updates the secondary variables at each node. The local search and intercommunication phases alternate, resulting in a cooperative search by the p nodes.

To facilitate a deeper understanding of the advantages and limitations of this class of algorithms, a theoretical foundation for these algorithms is developed using techniques from stochastic process theory and mathematical analysis. First, a theoretical basis for centralized evolutionary algorithms is specified in terms of construction and evolution of sampling distributions over the feasible space. Next, this foundation is extended to develop a general model of distributed coevolutionary algorithms. Convergence and convergence rate analyses are pursued for certain basic classes of objective functions, and analytical and simulation techniques are used

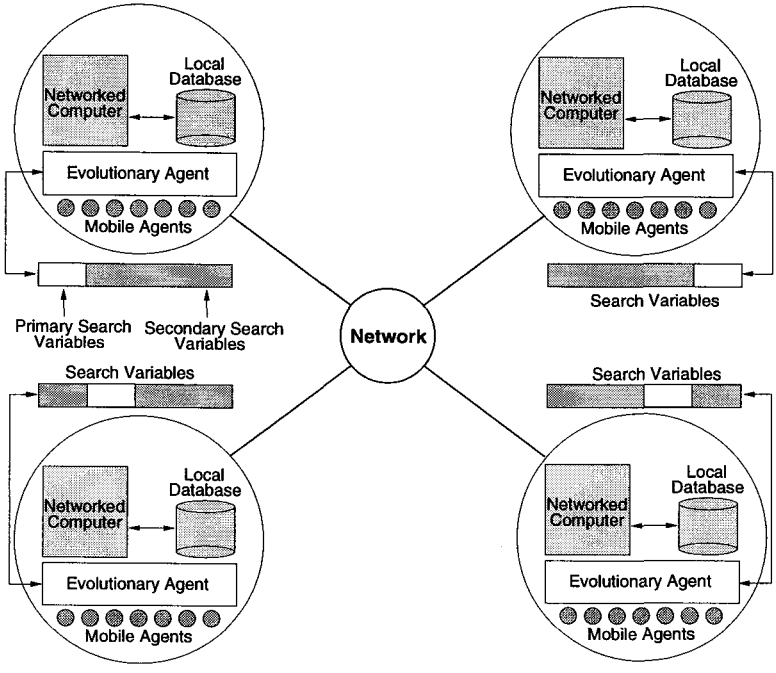


Fig. 1.2 Distributed coevolutionary computation.

to evaluate the performance of these algorithms when they execute in a network environment.

As a case study in distributed, network-based decision-making, an implementation and evaluation of the coevolutionary decision-making framework that is particularly suited to distributed network-enabled design and manufacturing organizations is presented. This implementation, the Coevolutionary Virtual Design Environment (CVDE), utilizes distributed evolutionary agents and mobile agents as principal entities that generate and execute queries among distributed applications and databases to support a global optimization of design, supplier, and manufacturing planning decisions for printed circuit assemblies. In this framework, an electronic interchange of design, supplier, and manufacturing information facilitates a concurrent, cooperative, network-efficient evolutionary search for promising planning alternatives. During the course of the evolutionary optimization, the CVDE generates *virtual designs* (complete integrated planning decisions) that are evaluated against an objective function based on cost and

time models. These computations require information collected dynamically over a network. As the evolution proceeds, successive generations of virtual designs are created, and the population systematically converges towards promising integrated planning decisions.

The methodology presented in this book can have a fundamental impact on the principles and practice of engineering in industrial product development in the network-based distributed environment that is emerging within and among corporate enterprise systems. In addition, the conceptual framework of the approach to distributed decision systems presented in this book may have much wider implications for network-based systems ranging from intelligent agent-based browser systems, to enhanced consumer and business services, and intelligent search techniques in scientific and commercial databases.

1.3 Principal Contributions

Towards the broad goal of systematic development of scalable and efficient evolutionary techniques for network-distributed decision-making, the principal contributions of this book are as follows:

- Problem formulation and analysis
  - Formal modeling of a class of coupled nonlinear assignment problems applicable to a variety of assembly oriented design-manufacturing domains, especially integrated design-supplier-manufacturing planning.
  - An analysis of the computational complexity of this problem class.
  - An evaluation of the applicability of alternative optimization algorithms to instances of this problem class, based on an analysis of the problem structure. This evaluation identifies this problem class as a good candidate for evolutionary optimization.
- Evolutionary optimization
  - Development of a theoretical foundation for evolutionary algorithms using techniques from stochastic process theory and mathematical analysis.
  - Extension of this framework to develop a general model of distributed coevolutionary algorithms applied to optimization

problems for which the variables are partitioned among p nodes.
  - Convergence and convergence rate analyses for basic classes of objective functions, and an evaluation of the performance of these algorithms when they execute in a network environment.
- Prototyping and evaluation of a network-based coevolutionary decision support system
  - Development of the CVDE that functions in a network-distributed environment and utilizes evolutionary agents and mobile agents as principal computational entities that facilitate a network-efficient global optimization in a highly coupled space of design, supplier, and manufacturing decisions.
  - Evaluation, and analysis of the CVDE for design-manufacturing decision support, and demonstration of the CVDE for printed circuit assembly planning.

1.4 Book Outline

Chapter 2 presents a review of the literature on topics underlying the research themes of the book. First, collaborative manufacturing is discussed. A description of competing deterministic and stochastic algorithms for combinatorial optimization is presented next. Next, an overview of evolutionary algorithm research is presented, followed by a discussion of certain special purpose evolutionary techniques useful in the distributed decision problem domain. Finally, overviews of the literature on agents and distributed problem solving are presented.

Chapter 3 first presents a formal model and computational complexity analysis of the problem of integrated design, supplier, manufacturing planning for modular products; the *design-supplier-manufacturing* problem. The general formulation is applicable to a variety of assembly-oriented design-manufacturing domains. The formulation is then adapted to model design-supplier-manufacturing planning for printed circuit assemblies. Next, an evaluation of the applicability of alternative optimization algorithms to this problem class is presented. This evaluation, based on the structure of the problem, supports the application of evolutionary optimization techniques.

Chapter 4 presents a theoretical foundation for a class of centralized evolutionary algorithms, wherein the evolution is described in terms of

construction and iterative progression of sampling distributions over the feasible space. This foundation is used to derive global convergence and convergence rate results for certain basic classes of objective functions.

Chapter 5 presents an extension of the theoretical foundation presented in the previous chapter to develop a general model of distributed coevolutionary algorithms applied to optimization problems for which the variables are partitioned among p nodes. Global convergence and convergence rate results similar to those in the previous chapter are developed for this class of evolutionary algorithms.

Chapter 6 presents an evaluation of the relative generational and network-based time performance of the centralized and distributed algorithms on several ideal objective function classes. Evolutionary techniques for the general design-supplier-manufacturing problem class are also presented.

Chapter 7 describes the architecture and implementation of the coevolutionary decision-making framework, CVDE, which utilizes evolutionary agents and mobile agents to support network-efficient global optimization of design, supplier, manufacturing planning decisions for printed circuit assemblies.

Chapter 8 presents a detailed evaluation of the centralized and distributed coevolutionary decision-making frameworks, identifies the principal factors that affect their network-based performance, and analyzes the requirements for applicability of the computational models to distributed network-enabled design and manufacturing organizations.

Chapter 9 presents a summary of the work, discusses topics for future research, and presents novel applications for the framework presented in this book.

Chapter 2

Background and Related Work

In this chapter, background information and a review of the literature on topics in collaborative manufacturing, combinatorial optimization, evolutionary algorithms, agents, and distributed problem solving are presented. Section 2.1 presents a review of the literature on collaborative manufacturing. In Section 2.2, a description of competing deterministic and stochastic algorithms for combinatorial optimization is presented. Section 2.3 reviews the literature on evolutionary algorithms. Section 2.4 presents background information on agents, and Section 2.5 reviews the literature in the field of distributed problem solving.

2.1 Collaborative Manufacturing

2.1.1 *Concurrent Engineering*

Traditionally, design and manufacturing of products are planned and executed with little or no interaction between the two phases. These decisions are chosen using the experience base in an organization and are often difficult to adapt to changing needs. *Concurrent engineering* is based on an argument that most of the life cycle cost of a product is determined at the design stage, and design choices significantly influence a product's manufacturability [Poli *et al.*, 1992]. Concurrent engineering stresses the importance of combining concerns of marketing, production, field service, and performance early in the design process, leading to multi-disciplinary collaborative design and manufacturing.

The Design for Assembly (DFA), Design for Manufacture (DFM), and Design for Manufacture and Assembly (DFMA) strategies were introduced by [Boothroyd and Dewhurst, 1983; Boothroyd, 1994]. DFM focuses on use

of selected materials and manufacturing processes for parts of an assembly, finding the most effective design, utilizing manufacturability as an important criterion. DFA consists of a set of rules, application of which leads to designs with fewer and simpler parts that are easier to orient, assemble, and disassemble. DFMA tools encourage interaction between design and manufacturing groups and are oriented mainly towards reduction of overall product cost. Current DFMA techniques do not consider lead times for parts procurement or manufacturing. They are batch-oriented tools using static databases and legacy models to provide design guidance.

It is reported by [Nevins *et al.*, 1989] that about 70% of the life cycle cost of a product is determined at its design stage, and they argue for a systematic early integration of design and manufacturing, and for including concerns of marketing, field service, and performance. Also, they emphasize advantages of achieving product variety through a combination of part options from product submodules. Such a combinatoric method of achieving model-mix production leads to a large number of product varieties by utilizing only a small number of options in each submodule. [He and Kusiak, 1997] consider the problem of designing low cost assembly systems for a family of modular products. Product varieties are achieved through component swapping among alternatives in a submodule, similar to the combinatoric approach of [Nevins *et al.*, 1989]. Their approach is specifically intended for production of customized modular products at low cost. Their heuristic optimization algorithm simultaneously considers assembly operation requirements and manufacturing scheduling issues [Cheng and Sin, 1990].

An optimization approach for design of products in a concurrent engineering framework is presented by [Dowlatshahi, 1992]. He focuses on design of mechanical products with parts that can be assembled. His algorithm involves modular decomposition of a product, establishing its feasible parts space through a combinatoric approach, reducing the number of feasible options, calculating utility values of design decisions, and finally optimization in the space of design decisions. He formulates the product design problem as a constrained integer linear program.

Engineous, a novel method for multidisciplinary integration of product design is described in [Tong *et al.*, 1992]. Their system utilizes genetic algorithms [Holland, 1994], expert systems, domain knowledge, object-oriented programming, and numerical optimization methods for integrated product design and engineering. Numerical optimization and expert systems are used to generate initial trial designs, and later genetic algorithms are

utilized to search the parameter space more thoroughly. Among other engineering design application examples, concurrent preliminary design of aircraft engine turbine blades is reported.

The problem of DFMA when competing manufacturing facilities are globally distributed and offer alternate process technologies for building a product is considered by [Taylor, 1997]. He poses the decision problem as a constrained integer linear program based on manufacturing costs including design costs, production costs, inventory costs, and transportation costs. He assumes availability of all costs via cost models.

2.1.2 *Agile Manufacturing*

The concept of "agility" was first introduced in 1991 by an industry-led group which observed that traditional manufacturing organizations were unable to keep up with rapid changes in the business environment [ame, 1991a; ame, 1991b]. The term *agile manufacturing* is used to characterize organizations that may frequently change to take advantage of new opportunities in the marketplace. Speed to market, ability to satisfy individual consumer or commercial customer preferences, and responsiveness to public concerns about social and environmental impacts of manufacturing are listed as determining factors of competitive advantage for an agile manufacturing environment [ame, 1991a]. The study [ame, 1991a; ame, 1991b] suggests that agile manufacturing systems favor smaller-scale modular production facilities where enterprises collaborate to contribute to new capabilities.

Recent research [Sanderson *et al.*, 1994; Hocaoğlu and Sanderson, 1996; Song and Nagi, 1997] has recognized the critical role of information infrastructure of design-manufacturing organizations in development of these agile capabilities, particularly in the context of more global and distributed organizations. [Sanderson *et al.*, 1994] introduced the concept of *multi-path agility* to improve productivity and response time using alternative resources and pathways accessed through improvements in information infrastructure. In this model of agility (Figure 2.1), improvement in throughput is not only achieved by shortening the response of individual entities on a single path, but also by selecting alternative routes to maximize the responsiveness of the entire process.

The Computer-Aided Manufacturing (CAMnet) project [Sobolewski and Erkes, 1995] presents enablers for delivering manufacturing services across virtual enterprises. Standards such as HTTP, HTML, and TCP/IP

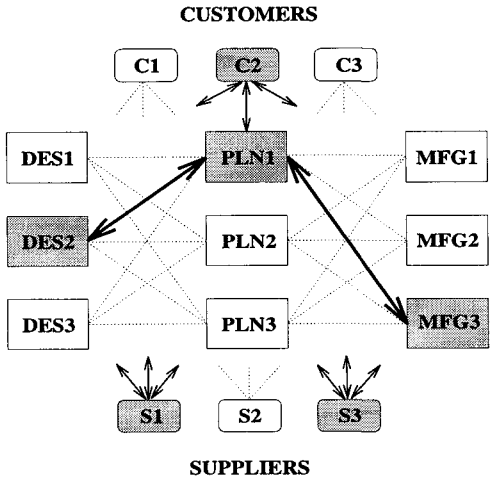


Fig. 2.1 Multi-path agility improves access to and selection of alternative resources.

are utilized for creating and delivering active documents to enhance collaboration between components of the virtual enterprise.

2.2 Combinatorial Optimization

Combinatorial optimization problems involve a search among a finite set of mutually exclusive alternatives, for those that minimize (resp. maximize) some desired criteria. Such problems appear very frequently in science, engineering, and management.

Definition 2.1 (Combinatorial optimization problem) *A combinatorial optimization problem is a pair* $(\mathcal{X}, \psi)$*, where:* (1) $\mathcal{X}$ *is a finite set of feasible points,* (2) ψ*, the cost function, is a mapping* $\psi : \mathcal{X} \longrightarrow \mathbb{R}$*, and the goal is to find an* $x_j \in \mathcal{X}$ *such that* $\psi(x_j) \leq \psi(x_i)\ \forall x_i \in \mathcal{X}$*. Such an* x_j *is called the global minimum.* ■

Often, finding the global minimum for a problem instance can be extremely time consuming since most combinatorial optimization problems are NP-Hard [Papadimitriou and Steiglitz, 1982]. However, it is often possible to find a solution that is better than others in a limited region of interest. This region of interest is called a neighborhood, and the solution in this region that is better than others is called a local minimum.

Definition 2.2 (Neighborhood) *Consider an instance $(\mathcal{X}, \psi)$ of an optimization problem. A neighborhood is a mapping $N : \mathcal{X} \longrightarrow 2^{\mathcal{X}}$.* ■

Definition 2.3 (Local minimum) *Consider an instance $(\mathcal{X}, \psi)$ of an optimization problem, a feasible point $x_i \in \mathcal{X}$, and a set $N(x_i)$. $x_j \in N(x_i)$ is a local minimum if $\psi(x_j) \leq \psi(x_k)\ \forall x_k \in N(x_i)$.* ■

Discrete choice problems incur a heavy penalty due to dimensionality, since the number of options grows explosively with the number of options along each dimension. This rapid and exponentially growing options space categorically rules out *brute-force* enumeration as a search technique for any other than problems of trivial size. Often, enumeration of only a tiny fraction of the options space is computationally feasible. Size of the search space is an issue only when algorithms use enumeration as a strategy. Unfortunately, for most combinatorial optimization problems, the known algorithms require some form of enumeration [Parker and Rardin, 1988].

Combinatorial optimization problems can be formulated as *integer programs*. This formulation scheme provides a uniform and convenient format for working with them. The general integer program (IP) formulation for a combinatorial optimization problem is:

$$\min_{1 \leq i \leq |\mathcal{X}|} \{\psi(x_i) \mid \mathbf{g}(x_i) = 0;\ x_i \in \mathbb{Z}_+^n\} \tag{2.1}$$

where the feasible set $\mathcal{X}$ is a constrained vector space of positive integers, and each $x_i \in \mathcal{X}$ is an n-vector of positive integers that satisfies an m-vector of constraint functions $\mathbf{g}(\cdot)$.

When the function to be minimized and the constraints are linear, the general integer program is reduced to an integer linear program (ILP):

$$\min_{1 \leq i \leq |\mathcal{X}|} \{c^T x_i \mid Ax_i = b;\ x_i \in \mathbb{Z}_+^n\} \tag{2.2}$$

where c is an n-vector of reals, A is an $m \times n$ matrix of reals, and b is an m-vector of reals.

Algorithms for integer programming can be broadly classified into two groups–*deterministic*, and *stochastic*. Deterministic algorithms use predictable and known search transitions, while stochastic algorithms incorporate probabilistic transitions. The class of deterministic algorithms can

be further classified into two subclasses–*exact*, and *approximate*. Exact deterministic algorithms are guaranteed to find the global minimum, whereas in the case of approximate deterministic algorithms, there is no guarantee of finding the global minimum. Stochastic algorithms on the other hand are approximate algorithms. Eventually, as the size of a problem grows, it becomes computationally infeasible to employ exact algorithms, and one is forced to turn to approximate algorithms, either deterministic or stochastic. This is the basis for the following discussion.

2.2.1 *Deterministic Algorithms*

This section discusses various exact and approximate deterministic algorithms applicable to integer programming problems.

2.2.1.1 *Exact Algorithms*

- **Linear Programming** (LP): A commonly used scheme for solving an ILP (2.2) using linear programming [Papadimitriou and Steiglitz, 1982; Goldfarb and Todd, 1989], allows the feasible set to assume positive real values and the solutions are rounded to the nearest integers. This technique is also known as solving an LP-relaxation of an ILP. This scheme is satisfactory when the variables assume large values and are not so sensitive to integer rounding. In other cases and especially when the variables can assume only binary values, this method can lead to solutions very far from optimal. However, in special situations when the matrix A is *totally unimodular*,[1] the optimal solution of the LP-relaxation is integral and is also the optimal solution to the ILP [Gondran and Minoux, 1984]. When the general IP (2.1) has a nonlinear cost function, a strategy is to approximate the nonlinearities using piecewise linear functions, making the IP amenable to solution via linear programming. [Papadimitriou and Steiglitz, 1982] describe various techniques for handling several types of nonlinearities in the cost function and constraint functions via mathematical reformulations so the IP can be solved using linear programming. Exploitation of the structure of the problem and intimate knowledge of the nature of the cost function are key components of all the schemes that uti-

[1] All square submatrices extracted from A have determinants equal to $0, 1$, or -1, which implies that all entries of A are elements of the set $\{0, 1, -1\}$.

lize a linear programming solver for an IP. Despite the possibility for various LP reformulations of an IP, exact solutions are generally possible only for small IPs and that too with a very heavy relative time penalty.

- **Cutting-Planes**: A popular strategy for achieving exact integral solutions to an LP-relaxation of an IP when the constraint matrix A cannot be cast in a totally unimodular form, is via cutting-planes [Papadimitriou and Steiglitz, 1982; Gondran and Minoux, 1984; Nemhauser and Wolsey, 1989]. A cutting-plane algorithm iteratively adds linear constraints to the feasible hyper volume identified by the LP solver. These hyper planes iteratively "cut away" portions of the feasible region so that integer feasible solutions are not excluded, and ultimately elements of the solution set from which the optimal solution is selected are all integral and feasible. However, a cutting-plane algorithm is efficient only in a limited class of IPs in which the constraint matrix A is "almost totally unimodular" (one or two rows of A have integral elements of any value).
- **Branch-And-Bound**: Branch-and-bound [Parker and Rardin, 1988; Nemhauser and Wolsey, 1989] is a classical technique for integer programming. This algorithm is based on the concept of "intelligently" enumerating feasible points of a combinatorial optimization problem without utilizing "brute-force." The method doesn't necessarily enumerate "all" points, but selects subtrees to search using bounds on possible outcomes. The feasible space is iteratively partitioned to yield subproblems, each subproblem is solved to obtain bounds on its cost function, subproblems whose lower bounds are higher than the known smallest upper bound are eliminated, promising subproblems are further partitioned, and the process of partitioning, bounds evaluation, and elimination or consideration is repeated until the best known lower bound does not show improvement. Branch-and-bound is a recursive strategy that is based on a tree search, where the nodes of a tree represent subproblems and the branches of a node are visited only if necessary. Branch-and-bound techniques frequently use LP solvers in order to compute bounds for subproblems and are able to guarantee optimality of a solution without exhaustively enumerating the feasible space. However, there are practical limitations to the use of this technique. Application of this tech-

nique requires that problems be easily decomposable with minimal coupling between subproblems. Also, in many instances integer feasible solutions are not readily available, making elimination of subproblems cumbersome, and in this case the algorithm fails as a result of explosive memory requirements [Lee and Mitchell, 1999]. Nevertheless, branch-and-bound remains a popular computational strategy for integer programming and the literature contains references to hybrid techniques (which improve on the basic technique) that utilize combinations of linear programming, branch-and-bound, and cutting-planes [Mitchell, 1999a; Mitchell, 1999b].

- **Dynamic Programming**: Dynamic programming [Bellman and Dreyfus, 1962; Papadimitriou and Steiglitz, 1982] is related to branch-and-bound to the extent that an "intelligent" enumeration of the feasible space is performed, but differs from branch-and-bound due to the strategy of working backwards from the final decision to the earlier ones. An application of this technique is limited to problems in which locally optimal decisions can be chained sequentially in order to generate globally optimal decisions, and to those problems that easily admit decomposition into well defined subproblems. In problems with multiple dimensions, dynamic programming runs into time and space problems due to the need to store an exponentially growing number of decision tables. Branch-and-bound has proved more effective than dynamic programming for many problem types and is thus more preferred.

2.2.1.2 *Approximate Algorithms*

When one is faced with optimization of an integer program with a complicated structure or of a large and practical size, seeking an exact solution may not be computationally feasible. In these practical instances, one is forced to consider approximate algorithms that can generate "high-quality" solutions in a "time-efficient" manner.

- All the exact optimization algorithms discussed in Section 2.2.1.1 can be utilized to generate approximate solutions, and as discussed earlier, this is inevitable for practical problem sizes.
- **Heuristics**: Heuristics [Pearl, 1984] are "tailored-made algorithms" based on rules that exploit the structure of a special

case of a given problem, and can often generate optimal solutions in *polynomial time* for these special cases. Moreover, these schemes can generate good solutions even for general problem instances of practical size. Some examples of popular heuristics are the Lin-Kernighan heuristic [Lin and Kernighan, 1973] for a symmetric traveling salesman problem,[2] the Kernighan-Lin heuristic [Kernighan and Lin, 1970] for graph partitioning, and the Christofides heuristic [Papadimitriou and Steiglitz, 1982] for the metric traveling salesman problem.[3] Some heuristics are applicable more generally–the "greedy heuristic" for instance, in which at each stage the best alternative among the feasible alternatives is chosen. The greedy heuristic has been applied to numerous combinatorial optimization problems, and works well for certain problems for which the structure is especially suitable to the strategy, but works very poorly on others [Parker and Rardin, 1988].

- **Tabu Search**: Tabu search [Glover, 1986; Hertz *et al.*, 1997] is gaining application for combinatorial optimization in spite of the fact that there is no known proof of its convergence, because it is an efficient optimization scheme for many problems. A Tabu search algorithm works by not only retaining information of the best solution detected, but also by systematically memorizing the itinerary through previous solutions. This memory helps restrict some search transitions in the neighborhood of a current solution, and thus discourages cycling among recently visited solutions. However, restrictions are relaxed when a solution has some preferred characteristics. The search selects the best solution from a set of feasible solutions that are subject to restrictions and relaxations. Later, the lists maintaining restriction and relaxation information are updated. The search is continued until some stopping criterion is met. Performance of a Tabu search algorithm is influenced significantly by a large number of parameters, and these need to be fine tuned for various problem domains. Also, maintaining and updating the search memory lists can be quite complicated and cumbersome. These drawbacks restrict the elegance of Tabu search as an optimization approach.

[2] A traveling salesman problem of n cities where the $n \times n$ distance matrix is symmetric.

[3] A traveling salesman problem where the arrangement of cities satisfies the triangle inequality.

2.2.2 *Stochastic Algorithms*

Evolutionary algorithms and simulated annealing are two popular stochastic algorithms applicable to integer programming problems. These algorithms do not rely on problem structure as much as most deterministic optimization schemes, and are more suitable in problem domains with cost functions having one or more of the following characteristics: time-variation, complexity, high dimension, nonlinearity, noise, nondifferentiablity. Due to these favorable characteristics and because they do not make strong assumptions regarding the cost function, these algorithms are easy to adapt to various problem domains and often obviate the development of intricate deterministic algorithms.

- **Evolutionary Algorithms**: This discussion is presented in Section 2.3.
- **Simulated Annealing**: Simulated annealing [Kirkpatrick *et al.*, 1983] is based on an analogy with the physical process of annealing, whereby a lattice structure of a solid is achieved by heating the solid to its melting point, and then slowly cooling it until it solidifies to a low-energy state. From the perspective of combinatorial optimization, simulated annealing works by randomly picking feasible solutions, improving on a solution by always accepting better-cost neighbors if they are selected, and allowing for a stochastically guided acceptance of worse-cost neighbors. The probability of acceptance of worse-cost neighbors decreases gradually, consistent with the analogy of gradual cooling of the heated solid. Simulated annealing, similar to evolutionary algorithms, has guaranteed asymptotic convergence behavior. In fact, a simulated annealing algorithm is a special case of a single-individual-evolutionary-algorithm that does not employ the recombination operator and uses a mutation operation for search. The most cited drawback of simulated annealing in combinatorial optimization, is speed of convergence. [Aarts *et al.*, 1997] present a summary of various speedup techniques in the literature.

2.3 Evolutionary Algorithms

Evolutionary Algorithms (EAs) include genetic algorithms [Goldberg, 1989; Holland, 1994], evolutionary programming [Fogel *et al.*, 1966;

Fogel, 1995], evolution strategies [Bäck, 1996], and genetic programming [Koza, 1992]. The principles of these related techniques define a general paradigm that is based on a simulation of natural evolution. EAs perform their search by maintaining at any time t a population $\mathcal{P}(t) = \{P_1(t), P_2(t), P_3(t), \cdots, P_p(t)\}$ of individuals. "Genetic" operators that model simplified rules of biological evolution are applied to create the new and desirably more superior population $\mathcal{P}(t+1)$. This process continues until a sufficiently good population is achieved, or some other termination condition is satisfied. Each $P_i(t) \in \mathcal{P}(t)$, represents via an internal data structure, a potential solution to the original problem. Choice of an appropriate data structure for representing solutions is very much an "art" than "science" due to the plurality of data structures suitable for a given problem. However, choice of an appropriate representation is often a critical step in a successful application of EAs, and effort is required to select a data structure that is compact, minimally superfluous, and can avoid creation of infeasible individuals. For instance, if the problem domain requires finding an optimal integer vector from the space defined by dissimilarly bounded integer coordinates, it is more appropriate to choose as a representation an integer-set-array[4] instead of a representation capable of generating bit strings.[5] Closely linked to choice of representation of solutions, is choice of a fitness function $\psi : \mathcal{P}(\cdot) \longrightarrow \mathbb{R}$, that assigns credit to candidate solutions. Individuals in a population are assigned fitness values according to some evaluation criterion. Fitness values measure how well individuals represent solutions to the problem. Highly fit individuals are more likely to create offspring by *recombination* or *mutation* operations. Weak individuals are less likely to be picked for reproduction, and so they eventually die out. A mutation operator introduces genetic variations in the population by randomly modifying some of the building blocks of individuals. Evolutionary algorithms are essentially parallel by design, and at each evolutionary step a breadth search of increasingly optimal subregions of the options space is performed. Evolutionary search is a powerful technique of solving problems, and is applicable to a wide variety of practical problems that are nearly intractable with other conventional optimization techniques. Evolutionary search schemes do not guarantee convergence to the global optimum in a predetermined finite time, but they are often capa-

[4] An integer-set-array is an array of bounded sets of integers.

[5] A representation that generates bit strings can create many infeasible individuals, and is certainly longer than a more compact sequence of integers. [Michalewicz, 1996] argues similarly.

ble of finding very good and consistent approximate solutions. Moreover, they are guaranteed to asymptotically converge under mild assumptions.

2.3.1 *Principal Techniques*

Evolution strategies, evolutionary programming, and genetic algorithms are generally considered the three principal evolutionary techniques in wide use, and are discussed below.

2.3.1.1 *Evolution Strategies*

Evolution Strategies (ESs) are suitable for optimization in continuous parameter spaces. ESs were developed to tackle parameter optimization problems, by Rechenberg and Schwefel in Germany during the 1960s [Bäck, 1996; Bäck and Schwefel, 1996]. ESs were first applied to hydrodynamical problems such as shape optimization of pipes, drag minimization of plates, and structure optimization of nozzles. The method employs a continuous representation and a mutation operator working on a single individual, to create a new individual. The better of the parent and offspring is selected to survive while the other is discarded. This simple strategy is known as the $(1+1)$-ES. Population oriented ESs are presented in detail in [Bäck, 1996; Bäck and Schwefel, 1996]. In the $(\mu+1)$-ES, $\mu > 1$ parents create one offspring using recombination and mutation operations. μ best survivors from the union of μ parents and 1 offspring are selected to constitute the next generation. This is equivalent to replacing the worst parent with the offspring if it is better than the worst parent. In the (μ, λ)-ES, where $\lambda > \mu \geq 1$, μ parents create λ offspring using recombination and mutation, and the best μ offspring deterministically replace the parents. It is possible that the best individual at generation $t+1$ is worse than the best individual at generation t. This policy of acceptance of temporary deterioration is sometimes useful to prevent convergence to a local optimum. In the $(\mu+\lambda)$-ES, μ parents create λ offspring using recombination and mutation, and the best μ survivors are chosen from the union of μ parents and λ offspring. This ES guarantees a monotone improvement in quality of solutions. (μ, λ)-ES and $(\mu+\lambda)$-ES are currently the commonly used evolution strategies. In both schemes, each individual in a population is a concatenation of a string of real valued object variables and a string of strategy variables. From a theoretical perspective, ESs have been shown to have asymptotic convergence behavior, and results on convergence rate are

available for simple test functions [Bäck, 1996].

2.3.1.2 *Evolutionary Programming*

Evolutionary Programming (EP) was developed by [Fogel *et al.*, 1966] in the United States during the 1960s. The original form of EP was designed for operation on finite state machines. However, modern application of EP has gravitated towards continuous parameter optimization, and currently EP competes with ESs for this application domain [Fogel, 1995]. The method works with a population $\mu > 1$ parents that generate μ offspring using a mutation operation. μ best survivors from the union of μ parents and μ offspring are selected to constitute members of the next generation. EP can be viewed as a special case of a $(\mu + \mu)$-ES when the recombination operation is disabled. Current EP methods are very similar to ESs, but with some differences in selection operations [Fogel, 1995; Bäck, 1996; Bäck and Schwefel, 1996], and as a result they have convergence results similar to that of ESs.

2.3.1.3 *Genetic Algorithms*

Genetic Algorithms (GAs) were developed by [Holland, 1994] in the United States during the 1970s, and significantly extended by [De Jong, 1975] and [Goldberg, 1989]. The original GA developed by Holland uses a binary representation of individuals and creates offspring primarily through recombination of genetic material from selected parent pairs; mutation is utilized with a low probability. Early popularity of a binary representation for GAs is due to Holland's *Schema Theory*, which tries to analyze GAs in terms of their expected schema sampling behavior. [Holland, 1994; Goldberg, 1989] provide detailed descriptions of the theory, but in short, the theory claims that short, low-order,[6] above average substrings receive exponentially increasing trials in subsequent generations. [Rudolph, 1994; Rudolph, 1996] has shown that regardless of the operations used, a GA that does not transfer the best solution in a given generation to the successive, will not converge, and the GA that performs this member transfer (also called *elitism*) will asymptotically converge. However, general results on convergence rates for GAs are not yet available. Recently, the signature binary string representation that distinguishes GAs from other EA methods has come under scrutiny. [Bäck and Schwefel, 1996] argue that a binary

[6] With only a few specified bit positions, and the others being "don't cares."

representation has serious disadvantages due to its propensity to further complicate an already nontrivial cost function by introducing multimodality. Alternate representations that are more apt for a problem domain have received considerable attention [Michalewicz, 1996].

2.3.2 *Theory and Applications*

The form of an EA that is preferred in this book is shown below.

Algorithm 2.1 (Evolutionary algorithm)
$t = 0$;
`initialize` $\mathcal{P}(t)$;
`while (termination condition != true) {`
  `evaluate` $\mathcal{P}(t)$;
  $\mathcal{P}(t)' =$ `select` $\mathcal{P}(t)$;
  $\mathcal{P}(t = t + 1) =$ `recombine` *and/or* `mutate` $\mathcal{P}(t)'$;
}

A discussion of the evolutionary operators follows.

2.3.2.1 *Evolutionary Operators*

Initialization for a EA can be done either randomly or seeding may be used to bias the search process. However, the latter initialization strategy may lead to poor convergence if the seeding algorithm generates very similar solutions for initial consideration [Burke *et al.*, 1998].

The evaluation process assigns to each $P_i(t) \in \mathcal{P}(t)$ a fitness score in the space of reals. In practice, the fitness function computation consists of three steps and is given by $\psi = \psi_f \circ \psi_s \circ \psi_o$, where $\psi_o : \mathcal{P}(t) \longrightarrow \mathbb{R}$ assigns an objective score (cost) to each $P_i(t) \in \mathcal{P}(t)$, $\psi_s : \mathbb{R} \longrightarrow \mathbb{R}$ is a function that *scales* objective scores, and $\psi_f : \mathbb{R} \longrightarrow \mathbb{R}$ is a fitness mapping.[7] As an EA search proceeds, the average fitness of a population may be almost equal to the member with the best fitness, and all members might have an equal chance of reproduction leading to a random search using average individuals [Goldberg, 1989]. Scaling methods such as *Linear Scaling* can offset this problem by maintaining the ability to discriminate between population members as the search proceeds. Other popular scaling methods are the

[7]Such a functional decomposition allows one to set fitness maximization as the goal regardless of the nature (maximization or minimization) of the underlying optimization problem.

Sigma Truncation method to deal with negative objective scores, and the *Power Law Scaling* method to manipulate search sensitivity [Michalewicz, 1996].

The role of the selection process is to encourage more fit individuals to reproduce and to discourage lesser fit individuals. Various selection schemes such as *Linear, Proportional, Truncation, Uniform* are described by [Blickle and Thiele, 1997], and are analyzed by [Chakraborty *et al.*, 1997].

Various recombination and mutation schemes for EAs are reported in [Goldberg, 1989; Bäck, 1996; Michalewicz, 1996]. A recombination operation allows offspring to receive portions of genetic information from various parents, while a mutation operation introduces new genetic material into the population. Formerly, EAs were seen as capable of uniform performance over a wide problem range. However, the more contemporary view acknowledges the fact that specific problems require customized setups for satisfactory performance. An EA's search performance is also significantly influenced by the values of its various parameters such as population size and probability of mutation. Parameter setting has received considerable attention in the EA community [De Jong, 1975; Grefenstette, 1986; Smith, 1997; Bäck, 1997; Spears, 1997]. Recently it has been demonstrated that parameter adaptation during the search based on the incorporation of expert knowledge is superior to static parameter setting and significantly improves convergence performance [Subbu and Bonissone, 2003].

2.3.2.2 *Theory*

Theoretical analyses of evolutionary algorithms have followed diverse tracks. The "Schema Theory" [Holland, 1994] approach represents an early attempt to explain behavior of GAs. [Srinivas and Patnaik, 1996] report the use of this theoretical foundation to derive results on GA behavior, and [Wright, 1999] has suggested an extension of the basic results from which Holland's result follows as a corollary. However, the theory has come under close scrutiny [Mühlenbein, 1997; Bäck and Schwefel, 1996], and it is argued that the theory due to Holland is unable to predict behavior of practical GAs. [Stephens and Waelbroeck, 1999] argue that in general there is no preference for short low-order schemata as is generally believed due to Holland's results.

The population of an EA depends purely on the state of the previous population, so tools from "Markov Chain" theory have found a natural application in modeling EA behavior [Davis and Principe, 1993; De Jong

et al., 1994; Spears and De Jong, 1996]. However, this approach suffers from the encapsulation of information at a very fine level of granularity complicating the derivation of useful predictive results.

A promising line of theoretical investigation has been the modeling of EAs as a class of global random search methods [Peck and Dhawan, 1995; Qi and Palmieri, 1994; Vose and Wright, 1995], which allows a more compact and elegant modeling of EAs. Some of the other approaches to modeling EA behavior are the "Statistical Mechanics" formulations of [Prügel-Bennett and Shapiro, 1994; Shapiro *et al.*, 1994], the "Order Statistics" formulation of [Bäck, 1995], and work due to [Grefenstette, 1995] that attempts to build predictive models based on the fitness distributions of evolutionary operators.

2.3.2.3 *Applications*

In spite of the fact that the last word in EA theory is yet to be written, EAs are finding increasing use in problem domains such as the generalized assignment problem [Wilson, 1997; Chu and Beasley, 1997], 0/1-knapsack problem [Raidl, 1998], scheduling [Deb and Chakroborty, 1998; Hart *et al.*, 1998], in manufacturing optimization [Joines *et al.*, 1996; Pierreval and Tautou, 1997], in nuclear reactor refueling planning [Bäck *et al.*, 1996], in DNA sequencing [Cedeño *et al.*, 1995], in engineering design optimization [Powell, 1990; Tong *et al.*, 1992], robot path planning [Hocaoğlu, 1997], and many others.

2.3.3 *Techniques for Constrained Optimization*

EAs are finding increasing application in mathematical programming domains, and in order to compete with conventional algorithms and to tackle hard practical problems, it is essential for EAs to be able to handle constraints introduced by the problem domain. If infeasible individuals can be generated, it is necessary to be able to systematically handle these situations. The major issues are: design of an objective function that can handle infeasible individuals, and preservation (or encouragement) of feasibility of solutions. In order to preserve solution feasibility it is possible to design special operators and data structures (for problem-representation) that only generate feasible solutions, as in [Schnecke and Vornberger, 1997] which presents a tree-based representation scheme for a VLSI placement problem. This generally intuitive idea is strongly supported in [Michalewicz,

1996]. Nevertheless, design of special purpose data structures that always generate feasible solutions can be quite difficult and is problem dependent, but from the perspective of combinatorial optimization problems, it is efficient to utilize data structures that always generate discrete values for parameters, and mostly generate feasible solutions. Another popular and simple technique for constraint handling is to eliminate infeasible individuals, as in evolution strategies. However, this technique is efficient only when the feasible space is a large portion of the entire search space [Michalewicz, 1996]. If a population consists of mostly infeasible individuals, it becomes necessary to improve (repair) individuals instead of rejecting them, otherwise the few feasible individuals would tend to dominate the search in later generations leading to premature convergence. For some problems such as the traveling salesman problem, it is considered computationally easy to repair individuals, while for some others such as the nonlinear transportation problem, repairing infeasible individuals might be as complex as solving the original problem [Michalewicz, 1996]. [Orvosh and Davis, 1993] propose a 5%-rule, which claims that an EA with a repair algorithm provides the best results when only 5% of the infeasible individuals are repaired. Penalty imposition on infeasible individuals is by far the most popular method to handle constraints. [Michalewicz, 1996; Michalewicz and Schoenauer, 1996] discuss a variety of penalty functions–static, dynamic, and adaptive–that can be applied. Design of penalty functions depends heavily on the problem domain and the type of results that can be achieved with them. However, some heuristic guidance for this design is available [Richardson *et al.*, 1989]. A proposal due to [Fonseca and Fleming, 1998a; Fonseca and Fleming, 1998b] is to consider constraints as "hard" goals and the cost function as a "soft" goal, and pursue a multicriteria optimization of the vector consisting of constraint violation measures and the cost function. The first requirement for such an optimization is satisfaction of all constraints, which is followed by an optimization of the original cost function. When not all goals can be simultaneously satisfied, they propose acceptance of the better of the infeasible solutions. [Barnier and Brisset, 1998] consider the integration of constraint satisfaction techniques with a GA for optimization of combinatorial problems whose search spaces are very large and complex and where many infeasible solutions can be generated. The GA is used to search in subspaces that satisfy all constraints. Feasible subspaces are identified using constraint satisfaction techniques. They demonstrate the method on vehicle routing and assign-

ment problems.

2.3.4 *Multi-Node Algorithms*

2.3.4.1 *Parallel and Distributed Algorithms*

Parallel and distributed implementations of evolutionary algorithms typically follow the *coarse-grained* approach of evolving independent populations on multiple nodes and occasionally migrating individuals between nodes, and the *fine-grained* approach of distributing individuals among multiple nodes and allowing localized interactions [Capcarrère *et al.*, 1999]. In these methods [Cantú-Paz, 1999; Matsumura *et al.*, 1998; Mühlenbein, 1991; Tanese, 1989], each node can potentially directly manipulate variables in all n dimensions. These models have been primarily pursued for speeding up computations in large-scale problems and for simultaneously alleviating the problem of premature convergence.

2.3.4.2 *Coevolutionary Algorithms*

Coevolutionary algorithms are distributed and consist of distinct distributed algorithm components that considered together follow various models of cooperation or competition. In this model, *different* subspaces of the feasible space are explored concurrently by the algorithm components. If a problem is such that the subproblem solved by each algorithm component is independent of the others (the problem is decomposable), then each algorithm component can evolve without regard to the other components. From an optimization perspective, in this case, each algorithm component optimizes in a landscape disjoint from the landscapes corresponding to the other algorithm components. However, many problems exhibit complex interdependencies, and from a coevolutionary perspective it is suggested that the effect of changing one of the interdependent subcomponents leads to a deformation or warping of the landscapes associated with each of the other interdependent subcomponents [Kauffman and Johnsen, 1991]. Recently, there has been a growing interest in applying coevolutionary systems for problem solving.

A coevolutionary distributed genetic algorithm for integrated manufacturing planning and scheduling is proposed by [Husbands *et al.*, 1997]. In this scheme, each species concentrates on identifying a feasible set of process plans for a single component to be manufactured. The species interact because they utilize shared manufacturing resources. Their individual fitness

functions take into account the need to utilize shared resources, and are based on various manufacturing costs. In order to resolve conflicts between species, an arbitrator species is simultaneously evolved. The fitness of the arbitrator species depends on its ability to resolve conflicts such that manufacturing delays are minimized. The individuals in each species compete internally in order to generate good process plans, and species compete at a higher level for shared manufacturing resources.

A coevolutionary model in which multiple species evolve independently, enter into temporary collaborations with certain members of the other species, and are rewarded based on the success of the collaboration in solving a problem is discussed in [Potter, 1997; Potter and De Jong, 2000; Ortega *et al.*, 1999]. A collaboration of all species is required to realize a coherent and complete problem solution. In this model, typically the best individual from each species is chosen as the representative that will collaborate with individuals of the other species. Thus, for evaluating the fitness of each individual in a given species, the best representatives from each of the other species are utilized to form the complete solution, following which the solution is evaluated. The fitness is assigned strictly to the individual being evaluated and is not shared with the representatives from the other species that participated in the collaboration. This "greedy" method of collaboration is proposed due to its simplicity. However, this simple pattern of interaction between species leads to entrapments in local optima. Other collaboration schemes that include random selections of representatives are also possible, and lead to better results [Potter, 1997].

A competitive coevolutionary model applicable to scheduling problems is presented by [Seredynski, 1997], and is based on game-theoretic models of limited interaction between individuals in competing populations. Individuals in a population have limited interaction with individuals in neighboring populations, and seek to maximize their fitness based on local evaluations. He demonstrates the successful emergence of global behavior achieved purely through local cooperation.

Recently, there has been development in the utilization of principles from Game Theory to model coevolutionary systems [Ficici *et al.*, 2000; Wiegand *et al.*, 2002]. While novel, their analyses require the key assumption of complete mixing of populations, which implies that during the evaluation phase of the algorithm, individuals in any given population are assumed to have interacted with all members of the other populations. For coevolutionary algorithms that execute over a distributed network environment, complete mixing of populations is not a viable option, since such

mixing would necessitate excessive network-based communication deleterious to the time-performance of the overall search compared to centralized evolutionary algorithms.

Coevolutionary approaches that follow a clearly adversarial model are based on the biological belief that an adaptive change of a species introduces a new challenge to the competing species, leading to an adaptive change on its part, and so on [Rosin, 1997]. In these systems, the fitness of an individual in a population is based on a competition with members from the other population. Rosin has applied the adversarial coevolutionary model to various problems, for instance, to the design of drugs that are robust across some drug resistance mutations, and to game playing.

2.3.5 *Techniques for Dynamic Environments*

Most optimization algorithms assume a static objective function, and this has generally been true of the research in evolutionary algorithms. However, many real-world applications are dynamic in nature, where it becomes essential to adapt solutions due to changes in the environment. Some examples of problems corresponding to dynamic environments are near real-time factory scheduling, resource allocation problems where resource characteristics change with time, and control problems where the system characteristics evolve with time. For these types of problems it is desirable to have available optimization algorithms that do not require restarts whenever environmental changes occur. Recently, evolutionary schemes have been explored for their applicability to such problems [Bäck, 1998; Grefenstette, 1999; Liles and De Jong, 1999; Trojanowski and Michalewicz, 1999; Weicker and Weicker, 1999]. Applicability analyses of evolutionary techniques to dynamic problems mostly have an experimental flavor, and the notion of adaptation to changes assumes more importance than convergence. When environmental changes occur, it is advantageous for the algorithm to maintain multiple alternative solutions rather than to have converged to a particular solution. Therefore, any technique for diversity maintenance in evolutionary algorithms can find useful application in dynamic problem contexts. For instance, introducing random solutions in a population or equivalently hyper-mutating some existing solutions has been found to be very useful [Grefenstette, 1992]. [Liles and De Jong, 1999] report on the use of speciation for diversity maintenance, and [Trojanowski and Michalewicz, 1999] report on the use of redundant genetic material that serves as a memory. In the latter scheme, an individual has active genetic material and a

limited memory (FIFO queue) that stores genetic material from its predecessors. An individual is first evaluated using its active genetic material and then reevaluated using genetic material from its predecessors. The best genetic material identified as a result of the evaluation process becomes the active genetic material for the next generation.

2.4 Agents

An *agent* [Russell and Norvig, 1995] is any module or system that has the ability to perceive its environment, and can select an action or action sequence to manipulate the environment. An agent has the ability to construct an internal representation of the environment, and uses reasoning to choose an action. Agents can be designed to act independently or collectively [Lesser, 1999]. Agents and agent-based systems have been around for a long time, but more recently they have increased in popularity mainly due to developments in distributed computation and the mainstream adoption of the *object-oriented* programming paradigm [Booch, 1994; Stroustrup, 1991], which provides a convenient and logically natural means to structure and construct agent-based systems.

An agent is considered *intelligent* if it can choose those actions that accomplish some predefined goal and simultaneously increase some performance measure. An agent is considered *autonomous* if it has the ability to choose actions based on its perception and experience rather than blindly follow a pre-programmed action schedule. Autonomy widens the scope of tasks that an agent can perform without any reprogramming. However, the task of designing autonomous agents is more challenging than designing non-autonomous agents. An agent is considered *mobile* if it has the ability to move itself in its environment. For certain agent applications, such as for a pipe inspection robot, mobility is important, while for others such as an electronic mail management system, mobility may not be necessary. The structure of an intelligent agent is shown in Figure 2.2.

Software agents [Nwana and Ndumu, 1997; Jennings *et al.*, 1998] are also sometimes known as *softbots* [Russell and Norvig, 1995]. An environment for a software agent essentially consists of information (computer files) and other software agents. Files may exist in repositories that are logically and physically distributed. Software agents can be located in the local memory of a single computer or they can exist in computers that are logically and physically distributed. Software agents have the ability to read (perception)

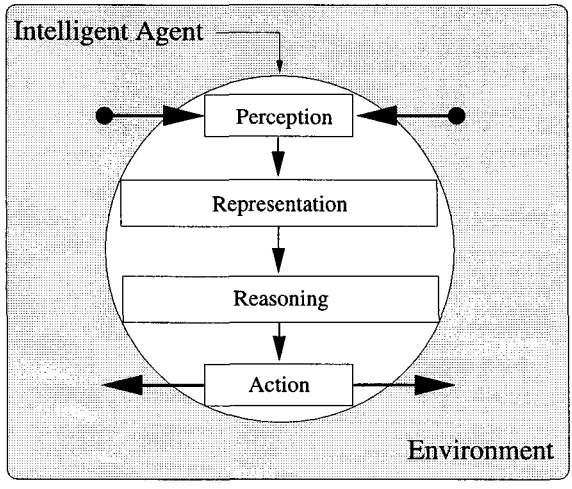


Fig. 2.2 Structure of an intelligent agent.

information from their environment and write (action) information to their environment. Their representation mechanism consists of data structures. Their reasoning mechanism consists of algorithms and data.

Research in the field of software agents and their applications is growing rapidly, and this includes descriptions of numerous prototype systems developed at various universities and research labs. Some have argued that given the highly multidisciplinary nature of research in agent-based systems, there is a tendency to view the literature in this fast growing field as chaotic and incoherent [Jennings *et al.*, 1998]. Given this, the aim is not to attempt to present a complete list of agent models and applications described in the literature. Instead, some important contributions are highlighted.

An excellent overview of agent-based systems, their essential characteristics, and applications is presented in [Hayes, 1999] . [Maes, 1994] describes the essential features of autonomous agents, and the common characteristics of agent-based solutions that have been proposed. [Jennings and Woolridge, 1998] describe prototype agent systems in information management, manufacturing, entertainment, process control, telecommunications, air traffic control and transportation systems. Recently, [Talukdar and de Souza, 1995] have proposed *A-Teams* (Asynchronous Teams) for problem solving. A-Teams are collections of autonomous agents that work iteratively and in parallel on populations of solutions. The motivation for A-Teams is derived

from collective problem solving observed in nature: for instance, ants in a colony cooperatively working towards construction of a nest. Such behavior is also seen in human societies: for instance, scientists in disparate fields collectively advancing scientific knowledge. Agents in A-Teams collaborate by modifying one another's results, which collect in shared memories. An A-Team agent consists of five components: an *input memory*, a *scheduler* for work triggering, a *selector* that chooses one or more solutions from the input memory, an *operator* that modifies the selected solutions and writes the result to the *output memory*.

2.5 Distributed Problem Solving

Distributed Problem Solving (DPS) is the cooperative solution of a problem by a group of loosely coupled[8] agents following a decentralized computational model. In this model, there are no centralized data stores, and no agent has enough information to make a complete decision; an agent requires assistance from other agents in the decision-making process. Also, agents may be physically and logically distributed over a computing environment. The problem is solved by intelligently combining subproblems solved by agents into an overall solution.

The fundamental areas of interest in DPS are the decomposition and coordination of computation among a society of agents so that structural demands of the task domain are matched [Chandrasekaran, 1981]. One of the earliest projects in DPS is the Contract-Net Protocol developed by [Smith, 1980; Smith and Davis, 1981], where computing nodes coordinate their activities through contracts. A "manager" node announces a task for which multiple eligible "contractor" nodes respond with bids. Contractor nodes receive pieces of the contract after a negotiation process, and in case a contractor requires assistance with its part of the problem, it assumes the role of a manager and subcontracts its part of the problem to other nodes. The original problem is solved in a top-down fashion by a network of contractor nodes. The method for task decomposition is specified a priori, and the Contract-Net framework is best suited to problems that can be hierarchically decomposed into nearly independent subtasks. [Lesser and Corkill, 1981] have proposed DPS systems that work effectively despite inconsistencies. In their "functionally accurate, cooper-

[8]Loosely coupled agents spend most of their time computing rather than communicating.

ative" approach, nodes cooperatively exchange and integrate partial and tentative results to construct a complete solution. Nodes make progress in problem solving using whatever information they can find. This work is motivated by the argument that consistency maintenance at all times is very expensive in practical systems. This model however leads to the possibility that agents might propagate and use incorrect partial results leading to unpredictable system performance. [Lesser, 1991] revisits the issue of functionally accurate, cooperative distributed problem solving and presents some techniques that may reduce the unpredictability of systems that propagate and use incorrect partial results. He proposes an increase in sophistication of local control in each agent so available information about a local search is more efficiently utilized, the exchange of meta-level information between agents so their local searches can be made while having a more global view, and satisficing control, in which less than optimal but acceptable levels of coordination between agents are used. [Sycara *et al.*, 1991] describe an architecture for solving distributed search problems using heuristics and constraint satisfaction methods, and apply it to decentralized job-shop scheduling.

Chapter 3

Problem Formulation and Analysis

3.1 Introduction

This chapter considers a class of distributed decision problems arising in integrated manufacturing planning. A formal model for this class of distributed decision problems is developed as a set of coupled nonlinear assignment problems.

Decisions made at the design stage have maximum impact on product cost and product realization time. These decisions affect choices of parts that are assigned to a design, selection of suppliers who will supply the parts, and selection of manufacturing resources that can produce the design. Design, supplier, and manufacturing decisions are not independent of each other and cannot be considered in isolation. Integrated planning methods thus assume importance as powerful means to improve competitiveness of products.

A formal model of integrated design, supplier, and manufacturing planning for modular products is developed. A decision problem in this class consists of three assignment problems (A_1, A_2, A_3). The assignment problem A_1 is the assignment of parts (from a parts library) to a design that satisfies a predetermined functional specification. Multiple designs (with dissimilar part assignments) that satisfy the functional specification are possible. For modular products, these designs can be generated combinatorially by combining parts that belong to module groups. The assignment problem A_2 is the assignment of suppliers (from a list of available suppliers) who will supply the parts in a combinatorially generated design, and the assignment problem A_3 is the assignment of designs to available manufacturing facilities. Each of these assignments affects the criteria of interest: typically cost and time for producing the product, and cannot be consid-

ered independent of one another. Also, the assignments have nonlinear (cannot be evaluated as weighted sums) effects on product cost, and product realization time. Given this problem structure, the goal then is to find optimal or near-optimal part, supplier, and manufacturing assignments at the product design stage.

Section 3.2 first presents a formal model of the integrated planning problem–the *design-supplier-manufacturing planning decision problem.* This formulation is applicable to a variety of assembly-oriented design-manufacturing domains where integrated design-supplier-manufacturing planning decisions are desired. An analysis of the computational complexity of this problem class is presented next. Section 3.3 first presents an adaptation of the formulation to model design-supplier-manufacturing planning for printed circuit assemblies, and then presents an analysis of the computational complexity of this problem. Section 3.4 presents an evaluation of the applicability of alternative optimization algorithms to the design--supplier-manufacturing planning problem class. This evaluation, based on the structure of the problem, supports the application of evolutionary optimization techniques.

This chapter is based in part on material that appears in the published literature [Subbu *et al.*, 1999], and earlier in [Subbu *et al.*, 1998a; Subbu *et al.*, 1998b; Subbu *et al.*, 1998c].

3.2 General Problem Formulation

$\mathcal{P} = \{P_1, P_2, P_3, \cdots, P_p\}$ is a non-empty set of parts. $\mathcal{C} = \{C_1, C_2, C_3, \cdots, C_c\}$, $(c \leq p)$ is a partition of $\mathcal{P}$. Therefore, the following properties hold:

$$\begin{aligned} &C_i \neq \emptyset \quad \forall i \\ &C_i \subseteq \mathcal{P} \quad \forall i \\ &C_i \cap C_j = \emptyset \quad \forall i \neq j \\ &\mathcal{P} = \bigcup_{i=1}^{c} C_i \end{aligned} \tag{3.1}$$

Each element of a C_i is *functionally equivalent*, and each C_i is an equivalence class called a part-equivalence-class (module type), and consists of functionally equivalent parts.

$\mathcal{M} = \{M_1, M_2, M_3, \cdots, M_m\}$ is a non-empty set of manufacturing re-

sources. $\mathcal{E} = \{E_1, E_2, E_3, \cdots, E_e\}, (e \leq m)$ is a partition of $\mathcal{M}$. Therefore, the following properties hold:

$$\begin{aligned} &E_i \neq \emptyset \quad \forall i \\ &E_i \subseteq \mathcal{M} \quad \forall i \\ &E_i \cap E_j = \emptyset \quad \forall i \neq j \\ &\mathcal{M} = \bigcup_{i=1}^{e} E_i \end{aligned} \tag{3.2}$$

Each element of an E_i is *functionally equivalent*, and each E_i is an equivalence class called a manufacturing-resource-equivalence-class, and consists of functionally equivalent manufacturing resources.

Let $\mathcal{D} = \{D_1, D_2, D_3, \cdots, D_d\}$ be the set of combinatorially generated designs, each consisting of non-empty subsets of parts, and let $\mathcal{S} = \{S_1, S_2, S_3, \cdots, S_s\}$ be the set of parts suppliers.

Parts are classified into equivalence classes, and parts within an equivalence class can functionally replace each other to realize a certain required function. Each part can be supplied by one or more suppliers. Satisfaction of a functional specification requires certain quantities of parts from one or more equivalence classes in order to realize a design that conforms to the specification. Multiple parts from the same equivalence class in "mixed and matched" quantities can be used as long as the required part quantity from that equivalence class is satisfied. There are also no restrictions on the suppliers who can supply these "mixed and matched" parts as long as they are available from these suppliers. The decision problem is then to determine these parts "mix and matches," the suppliers who will supply them, and the assignment of these parts to various manufacturing resources. Manufacturing resources are also classified into equivalence classes, and resources within an equivalence class are functionally equivalent. Parts from a part-equivalence-class have prescribed manufacturing resource requirements. There are no restrictions on how selected parts from a part-equivalence-class are assigned to manufacturing resources from a manufacturing-resource-equivalence-class, as long as the manufacturing specifications are adhered to.

Table 3.1 shows the variables used in the general problem formulation.

Table 3.1 Variables used in the problem formulation.

Variable	Description
$t_{ij} \in \{0,1\}$	$t_{ij} = 1 \Leftrightarrow$ an arc exists between P_i and S_j. $(i = 1,2,3,\cdots,p)$, $(j = 1,2,3,\cdots,s)$
$u_{ij} \in \{0,1\}$	$u_{ij} = 1 \Leftrightarrow P_i$ is a member of C_j. $(i = 1,2,3,\cdots,p), (j = 1,2,3,\cdots,c)$
$v_i \in \{0,1\}$	$v_i = 1 \Leftrightarrow$ an arc exists between C_i and $\mathcal{D}$. $(i = 1,2,3,\cdots,c)$
$w_i \in \{0,1\}$	$w_i = 1 \Leftrightarrow$ an arc exists between P_i and $\mathcal{D}$. $(i = 1,2,3,\cdots,p)$, and $w_i = \sum_{j=1}^{c} u_{ij} v_j$
$x_i \in \mathbb{Z}_+$	Functional specification that specifies instances of required elements from each C_i. $(i = 1,2,3,\cdots,c)$, and $x_i > 0 \Leftrightarrow v_i = 1$
$y_{ij} \in \{0,1\}$	$y_{ij} = 1 \Leftrightarrow M_i$ is a member of E_j. $(i = 1,2,3,\cdots,m), (j = 1,2,3,\cdots,e)$
$z_{ij} \in \{0,1\}$	$z_{ij} = 1 \Leftrightarrow$ an arc exists between C_i and E_j. $(i = 1,2,3,\cdots,c)$, $(j = 1,2,3,\cdots,e)$
$\phi_{ijk} \in \mathbb{Z}_+$	**Decision variable**, $(i = 1,2,3,\cdots,p)$ $(j = 1,2,3,\cdots,s), (k = 1,2,3,\cdots,m)$

$t_{ij} = 1$ implies P_i can be supplied by supplier S_j. $v_i = 1$ implies one or more part representatives from C_i are necessary for any design in $\mathcal{D}$ to be functionally consistent. $z_{ij} = 1$ implies part representatives selected from C_i must be assigned to manufacturing resources from E_j. $\phi_{ijk} > 0$ implies that many instances of part P_i are procured from supplier S_j and assigned to manufacturing resource M_k.

Each $D_i \in \mathcal{D}$ is written as a string

$$D_i = \left\langle (P_{11}^{(i)}, n_{11}^{(i)})(P_{12}^{(i)}, n_{12}^{(i)}) \cdots (P_{1|C_1|}^{(i)}, n_{1|C_1|}^{(i)}) \right\rangle \left\langle (P_{21}^{(i)}, n_{21}^{(i)})(P_{22}^{(i)}, n_{22}^{(i)}) \cdots (P_{2|C_2|}^{(i)}, n_{2|C_2|}^{(i)}) \right\rangle \cdots \left\langle (P_{c1}^{(i)}, n_{c1}^{(i)})(P_{c2}^{(i)}, n_{c2}^{(i)}) \cdots (P_{c|C_c|}^{(i)}, n_{c|C_c|}^{(i)}) \right\rangle$$

which is a concatenation of sequences of ordered pairs[1] representing elements of $\mathcal{P}$ and their instances. In an ordered pair $(P_{jk}^{(i)}, n_{jk}^{(i)})$, the superscript $^{(i)}$ is used to indicate the selection determined by design D_i, j refers to the equivalence class C_j, k is an internal index of the equivalence class C_j, $P_{jk}^{(i)} \in C_j$, and $n_{jk}^{(i)}$ denotes the number of instances of $P_{jk}^{(i)}$ in D_i. Also, $\forall D_i \in \mathcal{D}$

[1] Each sequence of ordered pairs is shown enclosed by the parentheses $\langle\ \rangle$. $|\cdot|$ represents the size of its argument.

$$P_{11}^{(i)}, P_{12}^{(i)}, \cdots, P_{1|C_1|}^{(i)} \in C_1 \; n_{11}^{(i)} + n_{12}^{(i)} + \cdots + n_{1|C_1|}^{(i)} = x_1$$

$$P_{21}^{(i)}, P_{22}^{(i)}, \cdots, P_{2|C_2|}^{(i)} \in C_2 \; n_{21}^{(i)} + n_{22}^{(i)} + \cdots + n_{2|C_2|}^{(i)} = x_2$$

$$\vdots \qquad\qquad\qquad\qquad \vdots$$

$$P_{c1}^{(i)}, P_{c2}^{(i)}, \cdots, P_{c|C_c|}^{(i)} \in C_c \; n_{c1}^{(i)} + n_{c2}^{(i)} + \cdots + n_{c|C_c|}^{(i)} = x_c$$

An example relationship structure between the sets $\mathcal{C}, \mathcal{D}, \mathcal{E}, \mathcal{M}, \mathcal{P}$, and $\mathcal{S}$ when applied to model the planning problem for a simple electronics assembly, at a certain level of abstraction, is shown in Figure 3.1. C_1 and C_2 are two classes of integrated circuits, and C_c is a power supply subassembly. E_1 is a class of integrated circuit placement machines, E_2 is a class of integrated circuit test machines, and E_e is a class of power supply test machines, all at a single manufacturing facility. Aggregate sets are shown in dotted lines, while arcs between elements are solid lines. Some of these relationships are annotated with variables from Table 3.1. For instance, $t_{11} = 1$, $t_{ps} = 1$, $t_{24} = 1$, $t_{7s} = 1$, $t_{2s} = 0$, $u_{11} = 1$, $u_{32} = 0$, $u_{7c} = 1$, $v_1 = 1$, $v_2 = 1$, $v_c = 1$, $y_{11} = 1$, $y_{21} = 1$, $y_{m2} = 1$, $y_{5e} = 1$, $z_{11} = 1$, $z_{12} = 1$, $z_{1e} = 0$, $z_{ce} = 1$.

3.2.1 *Constraints*

At this stage of the formulation, it is useful to list and describe the assignment constraints so the structure of the general problem emerges. (3.3), (3.4), (3.5), (3.6), (3.7) are the constraints.

$$\sum_{i=1}^{p} \sum_{j=1}^{s} (t_{ij} - 1)\phi_{ijk} = 0, \quad (k = 1, 2, 3, \cdots, m) \tag{3.3}$$

Constraint (3.3) restricts the decision variable so that a ϕ_{ijk} can assume a non-zero value only when there exists an arc between P_i and S_j. This constraint restricts the utilization of unavailable parts in combinatorially generated designs.

$$\sum_{i=1}^{p} (w_i - 1)\phi_{ijk} = 0, \quad \begin{cases} (j = 1, 2, 3, \cdots, s) \\ (k = 1, 2, 3, \cdots, m) \end{cases} \tag{3.4}$$

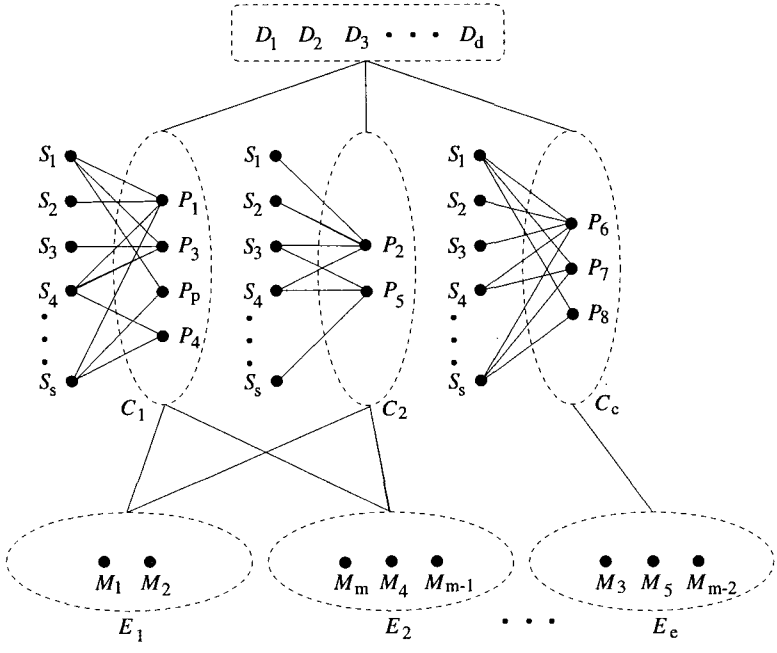


Fig. 3.1 An example relationship structure between the sets $\mathcal{C}, \mathcal{D}, \mathcal{E}, \mathcal{M}, \mathcal{P}$, and $\mathcal{S}$ when applied to model the planning problem for a simple electronics assembly, at a certain level of abstraction.

Constraint (3.4) restricts the decision variable so that a ϕ_{ijk} can assume a non-zero value only when there exists an arc between P_i and $\mathcal{D}$. This constraint prevents the utilization of those parts that belong to equivalence classes that contribute no part representatives to any design.

$$\sum_{\{i|u_{(i=1,2,3,\cdots,p)q=1}\}} \sum_{j=1}^{s} \sum_{\{k|y_{(k=1,2,3,\cdots,m)L=1}\}} \phi_{ijk} = x_q \qquad (3.5)$$
$$(q = 1, 2, 3, \cdots, c), L \in \{l|z_{q(l=1,2,3,\cdots,e)=1}\}$$

Constraint (3.5) restricts the decision variable such that the requirements imposed by the functional specification are adhered to. In this general formulation parts in a part-equivalence-class can be assigned to manufacturing resources in one or more manufacturing-resource-equivalence--classes. It is necessary to ensure that the number of parts (from a particular part-equivalence-class) assigned to manufacturing resources in a certain

manufacturing-resource-equivalence-class is consistent with the number of parts (from the same part-equivalence-class) assigned to manufacturing resources in a different manufacturing-resource-equivalence-class. For a given part-equivalence-class C_q, the index set $\{i\}$ of all its part members is identified via $\{i|u_{(i=1,2,3,\cdots,p)q=1}\}$. For this class C_q, $\{l|z_{q(l=1,2,3,\cdots,e)=1}\}$ is the index set of all manufacturing-resource-equivalence-classes such that there exists an arc between them and C_q. L is an element from this index set $\{l\}$. For any manufacturing-resource-equivalence-class E_L, the index set $\{k\}$ of all its elements is identified via $\{k|y_{(k=1,2,3,\cdots,m)L=1}\}$. The values of the decision variable ϕ_{ijk} are summed over the appropriate indices i, j, k. This sum is constrained to be equal to the functional specification x_q for part-equivalence-class C_q.

$$\sum_{\{k|y_{(k=1,2,3,\cdots,m)L_1=1}\}} \phi_{ijk} = \sum_{\{k|y_{(k=1,2,3,\cdots,m)L_2=1}\}} \phi_{ijk}$$
$$\begin{cases} (q = 1, 2, 3, \cdots, c), \{i|u_{(i=1,2,3,\cdots,p)q=1}\} \\ (j = 1, 2, 3, \cdots, s), L_1 \neq L_2 \\ L_1, L_2 \in \{l|z_{q(l=1,2,3,\cdots,e)=1}\} \end{cases} \tag{3.6}$$

Constraint (3.6) restricts the decision variable such that the assignment of parts to manufacturing resources does not violate manufacturing requirements. Intuitively, this constraint implies that if one were to select a certain quantity of a part (belonging to a particular part-equivalence-class) and assign it to manufacturing resources in a particular manufacturing-resource-equivalence-class, and if those part instances must also be assigned to manufacturing resources in some other manufacturing-resource-equivalence-class, then it is necessary that the assignments to manufacturing resources across manufacturing-resource-equivalence-classes stay consistent. The notation used to express this constraint is similar to that used in Constraint (3.5).

$$\phi_{ijk} \in \{0, 1, 2, \cdots\} \quad \begin{cases} (i = 1, 2, 3, \cdots, p) \\ (j = 1, 2, 3, \cdots, s) \\ (k = 1, 2, 3, \cdots, m) \end{cases} \tag{3.7}$$

Constraint (3.7) is for ensuring non-negativity and integrality of the decision variable.

The constraints in this general formulation provide significant flexibility in the assignment structure. In specific design-manufacturing situations with unique needs, additional constraints that restrict some of the flexibility can be added. The printed circuit assembly design, supplier, and manufac-

turing planning problem (see Section 3.3) is an example where additional constraints are added to the general set.

3.2.2 *Objectives*

The assignment constraints of the general design-supplier-manufacturing planning problem are described above. The assignments themselves may be evaluated using objective functions that are designed to provide estimates of the cost and time for producing a particular design. Global functions are intended to be estimates of various cost and time effects, and therefore do not represent exact measurements based on knowledge of a finite schedule.

Parts Cost

This estimates the total cost of parts in a design, and is given by

$$\pi = \sum_{q=1}^{c} \sum_{\{i|u_{(i=1,2,3,\cdots,p)q=1}\}} \sum_{j=1}^{s} \gamma_{ij}(\alpha_{ij}) \tag{3.8}$$

where

$$\alpha_{ij} = \sum_{\{k|y_{(k=1,2,3,\cdots,m)L=1}\}} \phi_{ijk}, \quad L = \underset{l}{\text{Any}}\{l|z_{q(l=1,2,3,\cdots,e)=1}\}$$

and $\Gamma(\cdot) = (\gamma_{ij}(\cdot)), (i = 1, 2, 3, \cdots, p), (j = 1, 2, 3, \cdots, s)$ is the parts cost function matrix. $\gamma_{ij}(\cdot)$ is an arbitrary linear or nonlinear function. $\gamma_{ij}(n)$ is the cost of n instances of part P_i when procured from supplier S_j.

Maximum Parts Lead Time

This estimates the lead time imposed by the part that incurs the maximum delay in procurement, and is given by

$$\mu = \max_{\substack{(q=1,2,3,\cdots,c) \\ \{i|u_{(i=1,2,3,\cdots,p)q=1}\} \\ (j=1,2,3,\cdots,s)}} \{\delta_{ij}(\beta_{ij})\} \tag{3.9}$$

where

$$\beta_{ij} = \sum_{\{k|y_{(k=1,2,3,\cdots,m)L=1}\}} \phi_{ijk}, \quad L = \underset{l}{\text{Any}}\{l|z_{q(l=1,2,3,\cdots,e)=1}\}$$

and $\Delta(\cdot) = (\delta_{ij}(\cdot)), (i = 1, 2, 3, \cdots, p), (j = 1, 2, 3, \cdots, s)$ is the parts lead time function matrix. $\delta_{ij}(\cdot)$ is an arbitrary linear or nonlinear function. $\delta_{ij}(n)$ is the lead time of n instances of part P_i when procured from supplier S_j.

Overhead Cost

This estimates any overhead costs associated with design and production, and is computed as a global function of the various assignments.

$$\sigma_o = f_{oc}([\phi]) \tag{3.10}$$

Overhead Lead Time

This estimates any overhead lead times that delay production, and is computed as a global function of the various assignments.

$$\tau_o = f_{ot}([\phi]) \tag{3.11}$$

Manufacturing Cost

This estimates the cost of production, and is computed as a global function of the various assignments.

$$\sigma_m = f_{mc}([\phi]) \tag{3.12}$$

Manufacturing Time

This estimates the various delay times for producing a design, which is dependent on factors such as transport, setup, and wait times, and is computed as a global function of the various assignments.

$$\tau_m = f_{mt}([\phi]) \tag{3.13}$$

The objectives described above present one model of reality, and in practice these would be customized to fit the production objectives of a particular company, product, and industry sector. Also, issues such as resource capacity are implicitly addressed through these objectives. For example, if a manufacturing resource is assigned more parts than its capacity, then rather than making this an infeasible production option, reality is better modeled by imposing time and cost penalties through the manufacturing objective functions. Another issue that deserves clarification is

that of manufacturing scheduling. The objective functions represent feedback mechanisms for high-level planning that do not explicitly consider manufacturing scheduling. However, the manufacturing cost, and delay time functions represent cost and time models that help estimate effects of scheduling.

3.2.3 *Optimization Problem*

It is desired to optimize the total cost and total delay time for producing a design. These are given by:

Total Cost

$$C^T = \pi + \sigma_o + \sigma_m \tag{3.14}$$

Total Delay Time

$$T^T = \max\{\mu, \tau_o\} + \tau_m \tag{3.15}$$

The optimization problem is written as the minimization of an aggregate function of constrained objectives (3.16).

$$\min \psi(C^T, T^T) \tag{3.16}$$

Subject To: Constraints (3.3), (3.4), (3.5), (3.6), (3.7).

The function $\psi(\cdot)$ in (3.16) represents a tradeoff between cost and time for producing a design, and assumes that the decision-maker is aware a priori of the relative importance of C^T and T^T.

3.2.4 *Complexity Analysis*

In order to evaluate the computational complexity of the problem class discussed in Section 3.2, some combinatorial results are established.

Result 3.1 *Let $C^{\mathrm{R}}(m,r)$ be the number of r-combinations of an m-set when repetition is allowed. This is also the number of ways to distribute r indistinguishable balls into m distinguishable cells, when cells can be empty.*

$$C^R(m,r) = C(m+r-1,r) = \binom{m+r-1}{r}$$

Proof: Proofs of Theorem 2.3 (p. 39), and Theorem 2.4 (p. 42) in [Roberts, 1984]. ■

Theorem 3.1 *Given r balls, r_1 of type 1, r_2 of type 2, $\cdots$, r_k of type k, with $r_1 + r_2 + \cdots + r_k = r$. Suppose that balls of the same type are indistinguishable, while balls of different types are distinguishable, and that there are m distinguishable cells. The number of ways to distribute the r balls into the m cells, when cells can be empty is*

$$\binom{m+r_1-1}{r_1}\binom{m+r_2-1}{r_2}\cdots\binom{m+r_k-1}{r_k}$$

Proof: The number of ways to distribute r_1 indistinguishable balls into m distinguishable cells, when cells can be empty, is (using Result 3.1) given by

$$\binom{m+r_1-1}{r_1}$$

Similarly, the number of ways to distribute r_2 indistinguishable balls into m distinguishable cells is given by

$$\binom{m+r_2-1}{r_2}$$

The number of ways in which these could happen together is

$$\binom{m+r_1-1}{r_1}\binom{m+r_2-1}{r_2}$$

Extending this argument, given r balls, the number of ways in which they can be distributed into m cells is given by

$$\binom{m+r_1-1}{r_1}\binom{m+r_2-1}{r_2}\cdots\binom{m+r_k-1}{r_k}$$

■

The computational complexity of the general *design-supplier--manufacturing planning decision problem* is presented below.

Consider an element $P_i \in \mathcal{P}$. If there exists an arc between P_i and S_j, part P_i can be supplied by supplier S_j, and the arc is called a suppliable-part option. It is assumed that any number of instances of a suppliable-part is possible. The number of arcs between P_i and elements of $\mathcal{S}$ is the number

of *distinct* suppliable-part options for P_i. This is given by

$$\sum_{j=1}^{s} t_{ij} \tag{3.17}$$

Consider an element $C_q \in \mathcal{C}$, which consists of some elements of $\mathcal{P}$. The total number of suppliable-part options in C_q, is derived using (3.17), and is given by

$$n_q = \sum_{\{i|u_{(i=1,2,3,\cdots,p)q=1}\}} \sum_{j=1}^{s} t_{ij} \tag{3.18}$$

Let x_q, the functional specification, specify the required number of parts from C_q. In order to satisfy the functional specification, it is possible to draw parts from one or more of the suppliable-part options in C_q. The number of ways in which parts can be drawn is obtained using Result 3.1. A mapping to the notation of Result 3.1 is: n_q is the m-set, and x_q is the r-combination. Thus, the number of ways in which parts can be drawn is given by

$$\binom{n_q + x_q - 1}{x_q} \tag{3.19}$$

In *a specific choice* of x_q elements, there can be repetitions of suppliable-parts. Let there be x_{q1} repetitions of suppliable-part 1, x_{q2} repetitions of suppliable-part 2, $\cdots$, x_{qn_q} repetitions of suppliable-part n_q, and $x_{q1} + x_{q2} + \cdots + x_{qn_q} = x_q$. Repetitions of a suppliable-part are considered indistinguishable, while the same part supplied by dissimilar suppliers corresponds to dissimilar suppliable-parts. Consider an element $E_l \in \mathcal{E}$ consisting of $|E_l|$ distinct manufacturing resources from $\mathcal{M}$. Assume that there exists an arc between C_q and E_l. So, x_q elements drawn from C_q can be assigned to manufacturing resources in E_l. The number of ways this assignment can be made is obtained using Theorem 3.1. A mapping to the notation of Theorem 3.1 is: x_q elements correspond to r, x_{q1} corresponds to r_1, x_{q2} corresponds to r_2, $\cdots$, x_{qn_q} corresponds to r_k, and $|E_l|$ manufacturing resources corresponds to m. Thus, the number of ways in which *a specific choice* of x_q elements can be assigned to the $|E_l|$ manufacturing

resources is given by

$$\begin{aligned}\zeta_{ql} &= \binom{|E_l| + x_{q1} - 1}{x_{q1}} \binom{|E_l| + x_{q2} - 1}{x_{q1}} \cdots \binom{|E_l| + x_{qn_q} - 1}{x_{qn_q}} \\ &= \prod_{i=1}^{n_q} \binom{|E_l| + x_{qi} - 1}{x_{qi}}\end{aligned} \tag{3.20}$$

If there are multiple arcs between a C_q and elements of $\mathcal{E}$, *a specific choice* of x_q elements can be together assigned to the manufacturing resources in ξ_q ways. This is derived using (3.20), and is given by

$$\xi_q = \prod_{\{l | z_{q(l=1,2,3,\cdots,e)=1}\}} \zeta_{ql} \tag{3.21}$$

The total number of ways in which x_q parts can be drawn from C_q and be assigned to manufacturing resources is derived using (3.19) and (3.21), and is given by

$$\binom{n_q + x_q - 1}{x_q} \xi_q \tag{3.22}$$

Given a complete functional specification $[x_1, x_2, \cdots, x_c]$, the **total number of coupled design-supplier-manufacturing decisions** is derived using (3.22), and is given by

$$\prod_{q=1}^{c} \left\{ \binom{n_q + x_q - 1}{x_q} \xi_q \right\} \tag{3.23}$$

which in an expanded form is

$$\boxed{\prod_{q=1}^{c} \left\{ \binom{n_q + x_q - 1}{x_q} \left(\prod_{\{l | z_{q(l=1,2,3,\cdots,e)=1}\}} \left[\prod_{i=1}^{n_q} \binom{|E_l| + x_{qi} - 1}{x_{qi}} \right] \right) \right\}}$$

3.3 Printed Circuit Assembly Problem

An adaptation of the general formulation is now presented. This adaptation models design-supplier-manufacturing planning for printed circuit assemblies.

Figure 3.2 shows a typical printed circuit assembly. Such an assembly consists of multiple types of parts mounted on a circuit board. There are three distinct requirements for producing a printed circuit assembly.

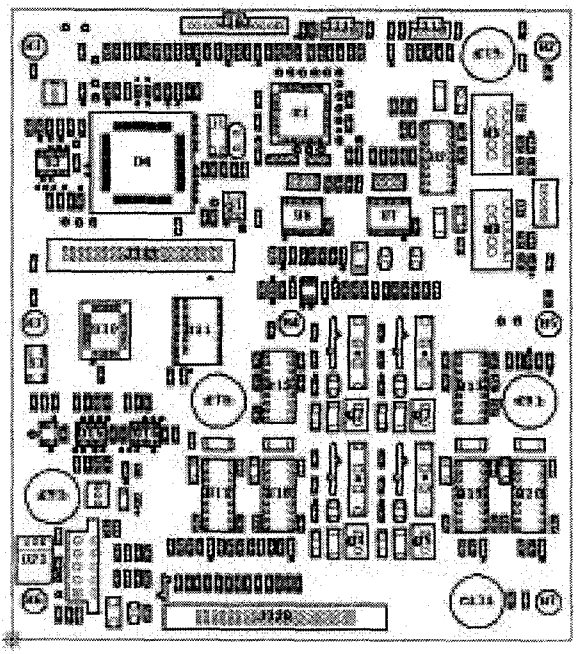


Fig. 3.2 Printed circuit assembly (courtesy of Pitney Bowes Inc.).

First, a string of parts (and their suppliers) needs to be specified along with a specification of the layout (and interconnections) of the parts on a printed circuit board. Next, a fabricator of the printed circuit board needs to be selected. Finally, a manufacturing facility that can assemble the product given a part string and circuit board, needs to be selected. Traditionally, these steps are executed sequentially with little or no interaction between the phases, and this results in numerous suboptimal iterations. The printed circuit assembly problem developed models the problem of integrated design-supplier-manufacturing planning for printed circuit assemblies.

The model presented is a restricted version of the general problem treated above. The first additional constraint is

$$|\mathcal{E}| = 2 \tag{3.24}$$

since the set of manufacturing resources consists of an equivalence class of printed circuit board fabricators and an equivalence class of printed circuit assembly facilities.

The second additional constraint is that manufacturing selections consist of a unique board fabricator and a unique assembly facility. For this a binary vector $A = (a_k), (k = 1, 2, 3, \cdots, m)$ is defined for convenience, where

$$a_k = 1, \text{ if } \sum_{i=1}^{p} \sum_{j=1}^{s} \phi_{ijk} > 0$$

$$a_k = 0, \text{ if } \sum_{i=1}^{p} \sum_{j=1}^{s} \phi_{ijk} = 0$$

This constraint is given by

$$\sum_{\{k|y_{(k=1,2,3,\cdots,m)L=1}\}} a_k = 1, \quad (L = 1, 2) \tag{3.25}$$

The third additional constraint is that parts can be procured from only one suppliable-part option in a part-equivalence-class.

$$\sum_{\{i|u_{(i=1,2,3,\cdots,p)q=1}\}} \sum_{j=1}^{s} \sum_{\{k|y_{(k=1,2,3,\cdots,m)L=1}\}} sgn(\phi_{ijk}) = v_q \tag{3.26}$$

where

$$(L = 1, 2), (q = 1, 2, 3, \cdots, c)$$

and

$$sgn(x) = \left\{ \begin{array}{l} 1 \text{ if } x > 0 \\ 0 \text{ otherwise} \end{array} \right\}$$

The overall objective function that is to be minimized is a heuristic weighting of the total cost and an exponential function of the total delay time. This follows the model shown in (3.16):

$$\psi(C^T, T^T) = C^T e^{(T^T - \alpha)/\beta} \tag{3.27}$$

where the overhead cost and time respectively embedded in C^T, T^T can be used to model printed circuit board fabrication cost and lead time. Also, α, β above are positive constants.

3.3.1 *Complexity Analysis*

The complexity result (3.2.4) is utilized to derive the complexity of the printed circuit assembly problem. Since constraints (3.24), (3.25), and (3.26) have been imposed on the general problem described in Section 3.2, some expressions in (3.2.4) become simpler.

Constraint (3.26) has the same combinatorial effect as setting $x_{q1} = x_q = 1$, and $x_{qi} = 0 \quad \forall i = (2, 3, \cdots, n_q)$. Therefore, (3.20) reduces to $\zeta_{ql} = |E_l|$. Due to constraints (3.24) and (3.25), (3.21) reduces to $\xi_q = |E_1||E_2|$, and in (3.23) ξ_q is moved outside the parenthesis. The complexity is given by

$$\begin{aligned} &\prod_{q=1}^{c} \left\{ \binom{n_q + x_q - 1}{x_q} \right\} \xi_q \\ &= \prod_{q=1}^{c} \left\{ \binom{n_q + 1 - 1}{1} \right\} |E_1||E_2| \\ &= |E_1||E_2| \prod_{q=1}^{c} n_q \end{aligned} \tag{3.28}$$

When n_q, the total number of suppliable-part options in a part-equivalence-class C_q, is at least 2, and $|E_1|, |E_2| \geq 2$, the problem size grows exponentially with the number of part-equivalence-classes and the numbers of manufacturing resources.

3.4 Algorithm Applicability Analysis

This section presents an evaluation of the applicability of alternative optimization algorithms to the design-supplier-manufacturing planning problem class. Given that the problem class has multiple variables, constraints, and nonlinear objectives, the goal is to select optimization algorithms that can tackle this broad class of integrated planning decision problems that are combinatorial in nature.

3.4.1 *Rationale*

Combinatorial problems incur a heavy penalty due to dimensionality, since the number of options grows exponentially with size of the inputs. Most combinatorial optimization problems are NP-Hard [Papadimitriou and Stei-

glitz, 1982], and this in general makes exact solution of their corresponding integer programs extremely time consuming. The rapid and exponentially growing options space categorically rules out exhaustive search for any other than small problems. Size of a search space is an issue only when algorithms use enumeration as a strategy. Unfortunately, for most combinatorial optimization problems, the known algorithms require some form of enumeration [Parker and Rardin, 1988]. Nevertheless, a variety of algorithmic strategies is applicable to optimization of integer programs. The algorithms that are capable of exact optimization intelligently exploit the structure of the problem, which is desirable. However, there are structural limitations to the applicability of exact optimization algorithms. The focus of this discussion is to evaluate applicability of these and other algorithms for the design-supplier-manufacturing planning decision problem class.

3.4.2 *Problem Structure*

An important first step in an evaluation of the applicability of alternative optimization algorithms to a specific problem class is an understanding of the characteristics of the problem class. This discussion first identifies the characteristics of the general design-supplier-manufacturing planning decision problem class, and then identifies the characteristics of the printed circuit assembly problem.

General Problem

Constraints (3.3), (3.4), (3.5), and (3.6) of the general problem model represent a set of linear equality constraints restricting assignment of the decision variable ϕ_{ijk} over the index space $(i = 1, 2, 3, \cdots, p)$, $(j = 1, 2, 3, \cdots, s)$, $(k = 1, 2, 3, \cdots, m)$. This linear equality constraint set can be rewritten in the form $A\phi = b$, where ϕ is a vectorization of the decision-variable vector, and has length $\ell_c = (p \cdot s \cdot m)$, A is an $\ell_r \times \ell_c$ matrix consisting of entries from the set $\{-1, 0, 1\}$, and b is an ℓ_r-dimensional column vector. Including constraint (3.7) defines a feasible space given by $\mathcal{X} = \{\phi : A\phi = b, \phi \in \mathbb{Z}_+^{\ell_c}\}$, which is convex.

The fact that the feasible space $\mathcal{X}$ is convex is an advantage from the perspective of exact optimization algorithms. Also useful is the property that elements of the matrix A are from the set $\{-1, 0, 1\}$, predisposing A to be totally unimodular, which is an advantage from the perspective of exact optimization algorithms. The objective function $\psi(\phi)$ consists of

coupled and nonlinear components. Due to couplings, a functional decomposition is difficult. Also since computation of this function requires cost and time information available from databases, it is difficult to predict a priori its nature over the search space. Therefore, the general objective is not guaranteed to be convex, differentiable, or continuous. The only practical method to compute an objective function value is to generate and evaluate an assignment of the decision variables. Due to these restrictions on the objective function it is not easy to perform any large-scale functional approximations.

Printed Circuit Assembly Problem

Since this problem is derived from the general problem model it shares the same characteristics described above. In addition, the introduction of constraints (3.25) and (3.26) introduces nonlinearities, which is a further complication from the perspective of any exact optimization algorithm.

3.4.3 *Evaluation of Alternative Algorithms*

Based on the problem characteristics discussed above, several alternative optimization methods are now systematically evaluated. The discussion below is based on the use of a single objective function $\psi(\phi)$.

Linear Programming

Linear Programs (LPs) are capable of exact optimization, and can be applied to solve problems such as $min\{c^T\phi : A\phi = b, \phi \in \mathbb{Z}_+^{\ell_c}\}$ under certain restrictive conditions. The constraint set requirement for an LP solution matches that of the general problem class. However, in order to apply an LP it is necessary that the cost vector c be available a priori. Another important feature is that a minimization of a linear combination $c^T\phi$ of the decision vector ϕ is performed. Though the objective of the general problem class is nonlinear, if it can be functionally described a priori, then it is conceivable that linear approximations can be generated towards the application of LPs. However, in the general problem class, neither the cost vector c can be specified a priori nor is it reasonable to obtain a functional description of the objective. These hurdles rule out the application of LP algorithms to the general problem class.

Branch-and-Bound

Branch-and-bound is general, intuitive, and can be applied to solve problems such as $z = min\{\psi(\phi) : \phi \in \mathcal{X}\}$ without the restrictions of LP. If the options space $\mathcal{X}$ can be decomposed into smaller sets such that $\mathcal{X} = \mathcal{X}_1 \cup \cdots \cup \mathcal{X}_K$, and $z^k = min\{\psi(\phi) : \phi \in \mathcal{X}_k\}$ for $k = 1, 2, 3, \cdots, K$, then $z = min_k z^k$. Application of branch-and-bound requires an algorithm that can generate good upper and lower bounds for a subproblem z^k. Traditionally, LP-based solvers are used for bounds generation, but for the general problem class this is not possible. Therefore, alternative algorithms are necessary for bounds generation. The second requirement for the application of branch-and-bound is an efficient strategy to decompose the feasible region $\mathcal{X}$, with the restriction that for moderate to large size problems it is computationally unattractive to generate and evaluate a large number of subproblems.

A branch-and-bound style divide and conquer optimization is possible for the general problem class, but the scheme cannot be used independent of a search algorithm that is capable of generating precise bounds for subproblems. Quality of the bounds is important for deciding which subproblems to prune and which to retain, and it is conceivable that an approximate algorithm used for bounds generation can lead to poor pruning choices that result in elimination of good subregions. Moreover, in the absence of any functional approximations to the objective, it is impossible to derive precise bounds for any subproblem, and this is an important hurdle faced in a consideration of branch-and-bound for the general problem class.

Deterministic Evaluative Search

Such a scheme can be compactly written as $\phi^{i+1} = D \cdot \phi^i$, where ϕ^i is a solution, ϕ^{i+1} is a successive solution, and D is a deterministic transition function that utilizes an evaluation $\psi(\phi^i)$ of ϕ^i. A variety of deterministic rules can be encoded in D, and this may include problem specific heuristics. Tabu search is an example algorithm that may be encoded in the transition function D. The algorithms in this class will result in approximate solutions unless they exploit some problem specific characteristic that is guaranteed to result in optimality. However, couplings among the decision variables and nonlinearity in the objective function in the general problem class complicate the identification of such characteristics. Moreover, there is no guarantee that a specific characteristic useful for a particular type of nonlinear objective will result in optimality for a broader range of

objectives.

Stochastic Evaluative Search

Such a scheme can be compactly written as $\phi^{i+1} = S \cdot \phi^i$, where ϕ^i is a solution, ϕ^{i+1} is a successive solution, and S is a stochastic transition function that utilizes an evaluation $\psi(\phi^i)$ of ϕ^i. The algorithms of this class generate approximate solutions, and are readily applied to the general problem class. Simulated annealing is an example algorithm that fits this description exactly. An evolutionary algorithm on the other hand may be written as $\langle \phi^{i+1} \rangle = S \cdot \langle \phi^i \rangle$ since it works with populations of solutions. Simulated annealing works with a single solution at any time, while evolutionary algorithms work with a population of solutions, and evolutionary algorithms therefore are more resistant to being trapped in local minima.

3.4.4 *Discussion*

Based on the discussion above, it is clear that the general problem class requires evaluative algorithms that by default generate approximate solutions, and do not make strong assumptions regarding the nature of the objective over the search space. Among the limited algorithm choices evolutionary techniques present most promise since they are population based, and are easily adaptable to various problems. Since evolutionary optimization is population based, there is a higher chance that multiple promising subregions of the space $\mathcal{X}$ will be simultaneously considered during the search. Their easy adaptability to problems increases their attractiveness as the optimization method of choice for this problem class. Deterministic and stochastic evaluative search methods can be creatively combined as *hybrid* search algorithms that are capable of good performance under certain conditions. For instance, an evolutionary algorithm could generate solutions that are improved by some deterministic heuristics that take advantage of certain characteristics of the problem domain, and such techniques can lead to improved solution quality and convergence.

Chapter 4

Theory and Analysis of Evolutionary Optimization

4.1 Introduction

The previous chapter focuses on developing a formal model and analyzing the computational complexity of the design-supplier-manufacturing planning decision problem class. This formulation is developed as a set of coupled nonlinear assignment problems, and is applicable to a variety of assembly-oriented design-manufacturing domains where integrated design-supplier-manufacturing decisions are desired. As an example, the formulation is adapted to model design-supplier-manufacturing planning for printed circuit assemblies. An evaluation of the problem structure supports the application of evolutionary techniques for optimization of the problems in this class.

This chapter focuses (in Section 4.2) on developing a theoretical foundation for modeling and convergence analysis of evolutionary algorithms applied to solve the general nonlinear problem

$$\max_{x \in \mathcal{X}} \psi(x) \tag{4.1}$$

where $\mathcal{X} \subset \mathbb{R}^n$ is a closed convex space of reals and $\psi : \mathcal{X} \longrightarrow \mathbb{R}_+$ is in general nonlinear and not separable. In general, it is acceptable to efficiently approximate the value $\psi^* = \max_{x \in \mathcal{X}} \psi(x)$, and find the corresponding global optimizer $x^* = \arg\max_{x \in \mathcal{X}} \psi(x)$, where $\psi^* = \psi(x^*)$. Though the end goal is application of evolutionary algorithms to combinatorial optimization problems, the theory is developed in the space of reals to facilitate a tractable and compact mathematical analysis.

The theoretical foundation describes evolutionary algorithms in terms of creation and evolution of sampling distributions over the feasible space.

Using this approach, global convergence and convergence rate results are derived for certain basic classes of objective functions (in Section 4.3).

This chapter is based on material that appears in [Subbu and Sanderson, 2003a], and earlier in [Subbu and Sanderson, 2000].

4.2 Theoretical Foundation

This section reviews the theory of *stochastic generational methods* proposed by [Zhigljavsky, 1991], which presents a convenient and intuitive mathematical foundation for modeling a class of evolutionary algorithms in terms of construction and evolution of sampling distributions over the feasible space $\mathcal{X}$. The key results in [Zhigljavsky, 1991] are adapted and presented without proof. Similar models are formulated and analyzed in the mathematical population genetics literature (see for example [Bürger, 1988; Karlin, 1979; Slatkin, 1970]). These general methods accommodate extensions to model other evolutionary algorithm variants. For instance, [Slatkin, 1970] has used this approach to study the effect of random mating, and [Peck and Dhawan, 1995] have formulated similar extensions to model genetic algorithms. The reader interested in a relative evaluation of this theoretical approach and other prevailing theoretical approaches is referred to [Peck and Dhawan, 1995], and to [Gao, 1998; Qi and Palmieri, 1994].

4.2.1 *Notation*

- μ_n is the Lebesgue measure on $\mathbb{R}^n$ or on $\mathcal{X} \subset \mathbb{R}^n$. Less formally, it is a general measure of volume in n dimensional real space.
- dx is compactly the infinitesimal volume element in n dimensional real space.
- $B(x, \epsilon) = \{z \in \mathcal{X} : \|x - z\| \leq \epsilon\}$, and $B(\epsilon) = B(x^*, \epsilon)$. $B(x, \epsilon)$ is the set of all points in $\mathcal{X}$ ϵ-close to x, and represents an enclosing ball of radius ϵ centered at x in the space $\mathcal{X}$. $B(x^*, \epsilon)$ represents the ϵ ball in $\mathcal{X}$ centered at the global optimizer x^*.
- $A(\epsilon) = \{z \in \mathcal{X} : |\psi(x^*) - \psi(z)| \leq \epsilon\}$ is the set of all points in the search space $\mathcal{X}$ that are ϵ-close to the global optimizer x^* in the objective space $\psi(\cdot)$.
- Given a set of events $\mathfrak{E}$, for any event $\mathfrak{e} \in \mathfrak{E}$ the image of the indicator function $\mathrm{I} : \mathfrak{E} \longrightarrow \{0, 1\}$ is given by $\mathrm{I}[\mathfrak{e}] = 1$ if $\mathfrak{e}$ is true,

and $\mathtt{I}[\mathfrak{e}] = 0$ if $\mathfrak{e}$ is false.

4.2.2 *General Algorithm*

A four-step scheme of a generational global stochastic search algorithm is presented. An evolutionary algorithm that does not include crossover but may include deterministic survival heuristics in combination with fitness proportional selection and mutation-oriented stochastic variation operations is a special case.

(1) **Initialization:** Set $g = 0$. Choose a probability distribution P_g on $\mathcal{X}$, and sample N times to obtain points $x_g^{(1)}, x_g^{(2)}, \cdots, x_g^{(N)}$ constituting the initial population.

(2) **Fitness Proportional Selection:** Construct the population distribution R_g on $\mathcal{X}$ by selecting N points based on their fitness. The probability of selecting a point with index j is given by

$$p_g^{(j)} = \frac{\psi(x_g^{(j)})}{\sum_{i=1}^{N} \psi(x_g^{(i)})} \tag{4.2}$$

(3) **One Step Evolution:** Construct the population distribution at the next generation P_{g+1} on $\mathcal{X}$ by varying R_g, and according to

$$P_{g+1}(dx) = \sum_{j=1}^{N} p_g^{(j)} Q_g(x_g^{(j)}, dx) \tag{4.3}$$

$Q_g(z, \cdot)$ represents a stochastic variational operator and is a generic function that can model a variety of mutation operations including deterministic survival heuristics. $Q_g(z, dx) = q_g(z, x)\mu_n(dx)$, and $\max_{z,x \in \mathcal{X}} q_g(z, x) < \infty$. $q_g(z, x)$ is an n dimensional probability density function representing the transition of point z to point x. Require that the width of such a function be non-zero, or alternatively, require its maximum height to be less than infinity. $Q_g(z, dx)$ represents the associated probability mass measured over the volume dx. Formally, $Q_g(z, \cdot)$ is a measurable, nonnegative function with respect to its first argument and a probability measure with respect to its second argument. A desired sample x in the distribution P_{g+1} is obtained by first sampling $R_g(dz)$ to obtain z, and then sampling $Q_g(z, dx)$. Stated al-

ternatively, a desired sample x in the distribution P_{g+1} is obtained by selection followed by stochastic variation.

(4) **Iteration:** Set $g \longleftarrow g+1$, and repeat from Step 2 until some termination criterion is met.

The construction of the distribution R_g on $\mathcal{X}$ via selection takes into account the global aspect of the search strategy, whereby a point z from all $\mathcal{X}$ is chosen, while the distribution $Q_g(z,\cdot)$, representing stochastic variation comprises the local aspect of the search strategy, whereby a point in the neighborhood of z is chosen. Importantly, the general form of $Q_g(z,\cdot)$ permits an encapsulation of deterministic problem-specific heuristics in addition to randomized heuristics, and may be used to some degree to offset the specificity of the assumption of fitness proportional selection above, though proportional selection is in and of itself not a restrictive assumption and not one that fosters only a narrow appeal or application. The nature of $Q_g(z,\cdot)$ largely determines the trade-off between search accuracy and search efficiency, and for a theoretical study it is convenient to select

$$Q_g(z,dx) = \frac{\phi((x-z)/\beta_g)\mu_n(dx)}{\displaystyle\int_{\mathcal{X}} \phi((y-z)/\beta_g)\mu_n(dy)} \tag{4.4}$$

where $\phi(\cdot)$ is decreasing for $\|x-z\| > 0$, is symmetrical, continuous, decomposes into a product of one-dimensional densities, $\beta_g > 0$, $\{\beta_g\}$ is a non-increasing sequence, and the denominator is a normalization constant. A random realization $x \in \mathcal{X}$ is obtained by repeatedly sampling $\phi(\cdot)$ until an ξ_g is obtained such that $z+\xi_g \in \mathcal{X}$, and then setting $x = z+\xi_g$. In this mode, the distribution $Q_g(z,\cdot)$ serves as a radially symmetric mutation operation. The above form is also preferred when there is random noise in fitness evaluations. In practice, and when there is no noise in the fitness evaluations, an option is to select $Q_g(z,\cdot)$ as

$$\begin{aligned} Q_g(z,A) = &\int_{\mathcal{X}} \mathtt{I}[x \in A : \psi(x) \geq \psi(z)] Q'_g(z,dx) + \\ &\mathtt{I}[z \in A] \int_{\mathcal{X}} \mathtt{I}[\psi(x) < \psi(z)] Q'_g(z,dx) \end{aligned} \tag{4.5}$$

where $Q'_g(z,dx)$ is also of the form of $Q_g(z,dx)$ as it appears in (4.4). What (4.5) represents is that if the probability mass of $Q_g(z,\cdot)$ is considered over an arbitrary set $A \subset \mathcal{X}$, then the contribution by the first integral is due

to the probability of sampling an $x \in A : \psi(x) \geq \psi(z)$, while the second integral contributes to the mass only when $z \in A$ and the sample $x \in \mathcal{X}$ is such that $\psi(x) < \psi(z)$. Essentially, expression (4.5) represents a policy by which the offspring survives only when its fitness is equal to or better than that of its parent. This expression demonstrates the incorporation of a deterministic survival heuristic based on parent-offspring competition. Other heuristics may be conceived and expressed as well using the above technique based on indicator functions.

4.2.3 *Basic Results*

This section presents four results based on the following assumptions:

(a) $0 < \psi(x) < \infty$ for all $x \in \mathcal{X}$.

(b) $Q_g(z, dx) = q_g(z, x)\mu_n(dx)$, and $\max_{z,x \in \mathcal{X}} q_g(z, x) < \infty$, as stated earlier.

(c) The global optimizer x^* is unique, and there exists an $\epsilon > 0$ such that $\psi(\cdot)$ is continuous in the finite region $B(\epsilon)$.

(d) There exists a $\delta > 0$ such that the sets $A(\epsilon)$ are connected for all $\epsilon : 0 < \epsilon < \delta$.

(e) For any $z \in \mathcal{X}$ and $g \to \infty$ the sequence of probability measures $Q_g(z, dx)$ weakly converges to the probability measure concentrated at the point z.

(f) For any $\epsilon > 0$ there exists a $\delta > 0$ and a natural number k such that $P_g(B(\epsilon)) \geq \delta$ for all $g \geq k$.

(g) $P_0(B(x, \epsilon)) > 0$ for all $\epsilon > 0, x \in \mathcal{X}$.

Result 4.1 *Let* **(a)**, **(b)** *hold, and let the probability distribution iteration follow the scheme in (4.3). Then, as* $N \to \infty$, *the distribution of the samples in the search follows the sequence*

$$P_{g+1}(dx) = \frac{\int_{\mathcal{X}} P_g(dz)\psi(z)Q_g(z, dx)}{\int_{\mathcal{X}} P_g(dz)\psi(z)} \tag{4.6}$$

□

Result 4.2 *Let* **(a)** *through* **(d)** *hold. Let* $\psi(\cdot)$ *be evaluated without random noise. Let* $Q_g(z, dx)$ *be chosen according to (4.5),* $Q'_g(z, dx) = q'_g(z, x)\mu_n(dx)$, *and let* $q'_g(z, x)$ *be an n-dimensional Gaussian density with independent coordinates whose variance is non-increasing with respect to*

generations, whereby $Q'_g(z, dx)$ weakly converges to the probability measure concentrated at the point z. Then, (**e**), (**f**), (**g**) *hold.* □

Result 4.3 *Let* (**a**) *through* (**d**) *hold. Let $\psi(\cdot)$ be evaluated either with finite bounded random noise or without noise. Let $Q_g(z, dx)$ be chosen according to (4.4), and let $q_g(z, x)$ be an n-dimensional Gaussian density with independent coordinates whose variance is non-increasing with respect to generations, whereby $Q_g(z, dx)$ weakly converges to the probability measure concentrated at the point z. Then,* (**e**), (**f**), (**g**) *hold.* □

Result 4.4 *Let Result 4.2 or Result 4.3 hold. Then, the distribution sequence (4.6) converges to the distribution concentrated at x^* as $g \to \infty$.* □

4.3 Convergence Analysis

This section presents convergence and convergence rate analyses of the evolutionary algorithm from Section 4.2 for unimodal and bimodal objectives based on Gaussians. A bimodal (multimodal in general) objective function is represented as a linear combination of Gaussians. Gaussian and linear combination of Gaussian objectives are used since they are representative of the basic function classes of interest in the analysis, they offer significant mathematical flexibility, and satisfy assumptions (**a**), (**c**), and (**d**) from Section 4.2.3. Moreover, a wide variety of objective functions with multiple modes may be readily constructed via linear combinations of Gaussians subject of course to suitable choices of the number of Gaussians, their respective scaling factors, and their respective means and covariances. Also, in the utilization of Gaussians for construction of objectives, there are no diagonality assumptions related to their covariance matrices. Non-diagonal covariance matrices in general introduce a nonlinear coupling between the search variables.

In the following discussion, the norm $\|\cdot\|$ is to be interpreted as the 2-norm $\|\cdot\|_2$.

4.3.1 *Convergence for a Unimodal Objective*

The theorem below mathematically describes the nature and evolution of the theoretical population distributions for a unimodal Gaussian objective

function given an initial distribution that is Gaussian and a Gaussian mutation operation that introduces stochastic variation.

Theorem 4.1 *Let $\psi(x) = \mathsf{N}[x^*, K]$, where K is positive definite, let the initial distribution also be a Gaussian $\mathsf{N}[x_0, C_0]$ with arbitrary mean x_0 and positive definite covariance C_0, and let the stochastic variation consist of an n-dimensional, zero-mean, coordinate-wise independent Gaussian mutation whose variance $W_g = \sigma_g^2 I$. Then,*

[a] *The population distribution is always Gaussian.*

[b] *The mean and covariance of the population distribution follow the respective iterations*

$$x_{g+1} - x_g = (I + KC_g^{-1})^{-1}(x^* - x_g)$$
$$C_{g+1} = W_g + (K^{-1} + C_g^{-1})^{-1}$$

[c] *C_g is always positive definite.*

Proof: [a] Algebraic manipulation of (4.6) using the assumptions in this Theorem leads to the probability distribution iteration $P_{g+1}(dx) = \mathsf{N}[(K^{-1} + C_g^{-1})^{-1}(K^{-1}x^* + C_g^{-1}x_g), W_g + (K^{-1} + C_g^{-1})^{-1}]\mu_n(dx)$ (see Appendix A.1), which establishes that the population distribution is always Gaussian.

[b] The expression $x_{g+1} = (K^{-1} + C_g^{-1})^{-1}(K^{-1}x^* + C_g^{-1}x_g)$, which governs the evolution of the mean of the distribution, can be rewritten as $x_{g+1} - x_g = (I + KC_g^{-1})^{-1}(x^* - x_g)$. The expression governing the evolution of the covariance of the distributions is evident.

[c] Since K and C_0 are positive definite, $K^{-1} + C_0^{-1}$ and $(K^{-1} + C_0^{-1})^{-1}$ are positive definite. By definition, W_g is positive semi-definite, and so $C_1 = W_0 + (K^{-1} + C_0^{-1})^{-1}$ is positive definite. Continuing in this fashion, it can be shown by induction that C_g is positive definite. ■

The analysis is not restricted to normally distributed initial populations. For instance, uniform initial population distributions may be utilized as well without any loss of generality. Gaussian initial population distributions are selected since their use highlights in a mathematically elegant manner the evolution of the mean of the sampling distribution to the optimum. However, from a practical perspective, a Gaussian distribution with large standard deviations may be easily used to approximate a uniform distribution.

The discussion following (4.4) lists the required properties of the mutation operation, and these properties are indeed quite broad. Gaussian

mutation fits quite nicely into this framework, and keeps the mathematics elegantly tractable.

The next theorem describes the rate of convergence to the global optimum of the mean of the theoretical population distribution for a unimodal Gaussian objective under the assumption that the initial population distribution is coordinate-wise independent and has a uniform standard deviation. The second part of the theorem shows that as the variance of the mutation vanishes the spread of the population distribution reduces to zero.

Theorem 4.2 *Let Theorem 4.1 hold, and let* $C_0 = cI, c > 0$. *Then,*

[a] *The sequence* $\{x_g\}$ *converges geometrically to* x^*.
[b] *If* $W_g = 0$ *then* $C_g \to 0$ *as* $g \to \infty$.

Proof: [a] The iteration $x_{g+1} - x_g = (I + KC_g^{-1})^{-1}(x^* - x_g)$ can be rewritten as $x_{g+1} - x^* = [I - (I + KC_g^{-1})^{-1}](x_g - x^*)$. Since K, C_g^{-1} are positive definite, KC_g^{-1} has only positive eigenvalues (Theorem 6.2.3 [Ortega, 1987]). Then all eigenvalues of $I + KC_g^{-1}$ are larger than unity, and $I + KC_g^{-1}$ is nonsingular. So, the eigenvalues of $(I + KC_g^{-1})^{-1}$ are all in the range $(0, 1)$, and the eigenvalues of $[I - (I + KC_g^{-1})^{-1}]$ are also in the range $(0, 1)$. It can be shown by induction that if $C_0 = cI, c > 0$ then $[I - (I + KC_g^{-1})^{-1}]$ is always positive definite (see Appendix A.2). The computation generating the sequence $\{x_g\}$ is viewed as a mapping $T : \mathcal{X} \longrightarrow \mathcal{X}$, where $x_{g+1} = T(x_g)$, and

$$\begin{aligned} \|x_{g+1} - x^*\| &= \left\|[I - (I + KC_g^{-1})^{-1}](x_g - x^*)\right\| \\ &\leq \left\|I - (I + KC_g^{-1})^{-1}\right\| \|x_g - x^*\| \end{aligned}$$

Let $A_g = I - (I + KC_g^{-1})^{-1}$. $\|A_g\| = \rho(A_g) \in (0, 1)$, where $\rho(A_g)$ is the spectral radius of A_g. Therefore, $\|x_{g+1} - x^*\| \leq \alpha_g \|x_g - x^*\|$, where $\alpha_g = \rho(A_g)$. Since α_g varies with generations, choose $\alpha = \max_g \{\alpha_g\}$, and have $\|x_{g+1} - x^*\| \leq \alpha \|x_g - x^*\|, \alpha \in (0, 1)$. Then, the computation $T(\cdot)$ is a pseudo-contraction [Bertsekas and Tsitsiklis, 1997] with modulus α in the *worst case*, and converges geometrically to the fixed point x^*.

[b] Let g be the generation from which point on $W_g = 0$. Then, $C_{g+1} = (K^{-1} + C_g^{-1})^{-1}$, and by induction $C_{g+m} = (mK^{-1} + C_g^{-1})^{-1} = K(mI + C_g^{-1}K)^{-1}$. For fixed K, C_g, the diagonals in $(mI + C_g^{-1}K)$ tend to ∞ as $m \to \infty$, while the off-diagonals are fixed. Therefore $(mI + C_g^{-1}K) \approx mI$ for large m, and $C_{g+m} \approx K(mI)^{-1} = \frac{1}{m}K$. Then, $\lim_{m\to\infty} C_{g+m} = 0$. ■

4.3.2 *Convergence for a Bimodal Objective*

The theorem below mathematically describes the nature and evolution of the theoretical population distributions for a bimodal Gaussian-based objective function given an initial distribution that is Gaussian and a Gaussian mutation operation that introduces stochastic variation. The bimodal Gaussian is a linear combination of Gaussians, and once again, the Gaussian assumptions for the initial distribution and mutation are not restrictive.

Theorem 4.3 *Let* $\psi(x) = \omega_a \mathsf{N}[x_a^*, K_a] + \omega_b \mathsf{N}[x_b^*, K_b]$, *where* K_a, K_b *are positive definite,* $\omega_a, \omega_b > 0$, *let the initial distribution also be a Gaussian* $\mathsf{N}[x_0, C_0]$ *with arbitrary mean* x_0 *and positive definite covariance* C_0, *and let the stochastic variation consist of an n-dimensional, zero-mean, coordinate-wise independent Gaussian mutation whose variance* $W_g = \sigma_g^2 I$. *Then,*

[a] *The population distribution is always a linear combination of Gaussians.*

[b] *The mean and covariance of two principal population distribution components in each generation follow the respective iterations*

$$x_{(i,g+1)} - x_{(i,g)} = (I + K_i C_{(i,g)}^{-1})^{-1}(x_i^* - x_{(i,g)})$$
$$C_{(i,g+1)} = W_g + (K_i^{-1} + C_{(i,g)}^{-1})^{-1}$$

where $C_{(i,0)} = C_0, i = (a, b)$.

[c] $C_{(i,g)}$ *is always positive definite.*

Proof: [a] Algebraic manipulation of (4.6) using the assumptions in this Theorem, and the techniques from Theorem 4.1a, directly leads to this conclusion.

[b] This conclusion follows directly from the algebraic expressions in [a], which are similar to the expressions in Theorem 4.1b.

[c] This conclusion is drawn using the techniques from Theorem 4.1c.■

The next theorem describes the rate of convergence to the global optimum of the mean of a principal population distribution component for a bimodal Gaussian-based objective under the assumption that the initial population distribution is coordinate-wise independent and has a uniform standard deviation. The second part of the theorem shows that as the variance of the mutation vanishes the probability mass in the epsilon region around the global optimum approaches one, implying convergence of the population distribution.

Theorem 4.4 *Let Theorem 4.3 hold, and let* $C_0 = cI, c > 0$. *Then,*

[**a**] *The sequence* $\{x_{(i,g)}\}$ *converges geometrically to* x_i^*.
[**b**] *If* $W_g = 0$ *then* $C_{(i,g)} \to 0$ *as* $g \to \infty$.
[**c**] *If* $W_g = 0$ *and the global optimizer is unique then* $P_g(B(\epsilon)) \to 1$ *as* $g \to \infty$.

Proof: [**a**], [**b**] These conclusions are drawn using the respective techniques from Theorems 4.2a,and 4.2b.

[**c**] Based on the given conditions the requirements for Result 4.3 hold, and therefore the conclusion follows from an application of Result 4.3 and Result 4.4. ■

In the above results, large sample assumptions are made, and a practical algorithm with a finite sample size is seen as an approximation to the algorithm with a large sample assumption. It is important to note that such an assumption facilitates a more tractable mathematical analysis while simultaneously providing a quantitative framework for describing average behavior of the algorithm that an algorithm with finite sample size approximates.

In the above results, large sample (infinite population) assumptions are made, and a practical algorithm with a finite sample size is seen as an approximation to the algorithm with a large sample assumption. It is important to note that such an assumption facilitates a more tractable mathematical analysis while simultaneously providing a quantitative framework for describing average behavior of the algorithm that an algorithm with finite sample size approximates. Gaussian-based objective functions are utilized to derive convergence and convergence rate results, and the linear combination of Gaussians method may be utilized to readily construct a wide variety of multimodal objectives. The bimodal Gaussian-based objective function, an instance of a multimodal Gaussian-based function, is used to derive convergence and convergence rate results for a function more complex than with one mode. The reader would notice that the techniques from Theorems 4.3 and 4.4 may be readily extended to Gaussian-based objectives with more than two peaks.

Chapter 5

Theory and Analysis of Distributed Coevolutionary Optimization

5.1 Introduction

The previous chapter focuses on developing a theoretical foundation for modeling and convergence analysis of centralized evolutionary algorithms applied to optimization problems. This theoretical foundation describes evolutionary algorithms in terms of creation and evolution of sampling distributions over the feasible space. Using this approach, global convergence and convergence rate results are derived for certain basic classes of objective functions.

This chapter focuses on extending the theoretical foundation from the previous chapter to develop a general model of distributed coevolutionary algorithms. Section 5.2 develops such a model for algorithms applied to optimization problems (5.1) for which the variables are partitioned among p nodes.

$$\max_{x \in \mathcal{X}} \psi(x) \tag{5.1}$$

$\mathcal{X} \subset \mathbb{R}^n$ is a closed convex space of reals and $\psi : \mathcal{X} \longrightarrow \mathbb{R}_+$ is in general nonlinear and not separable. In general, it is acceptable to efficiently approximate the value $\psi^* = \max_{x \in \mathcal{X}} \psi(x)$, and find the corresponding global optimizer $x^* = \arg\max_{x \in \mathcal{X}} \psi(x)$, where $\psi^* = \psi(x^*)$. The variable vector x is partitioned into p blocks $(x_1, x_2, \cdots, x_p)$, where $x_i \in \mathbb{R}^{n_i}, n = \sum_{i=1}^{p} n_i$, and the p blocks are distributed among p nodes. For a certain problem (5.1), a distribution of the variables into p blocks is guided by locality of information required for function evaluations at the computational nodes.

Given a feasible space $\mathcal{X}$ and a variable distribution, each node i performs a local evolutionary search in its primary subspace $\mathcal{X}_i$, while the

variables corresponding to the secondary subspaces at a node are clamped (see Figure 5.1). An intercommunication operation updates the respective secondary variables at all nodes. Following this, the local search proceeds using updated information, and in this fashion the local and global operations of the distributed search alternate, resulting in a cooperative search. The search space $\mathcal{X}$ in this model of computation is therefore the product space of the p subspaces, and is given by $\mathcal{X} = \prod_{i=1}^{p} \mathcal{X}_i$.

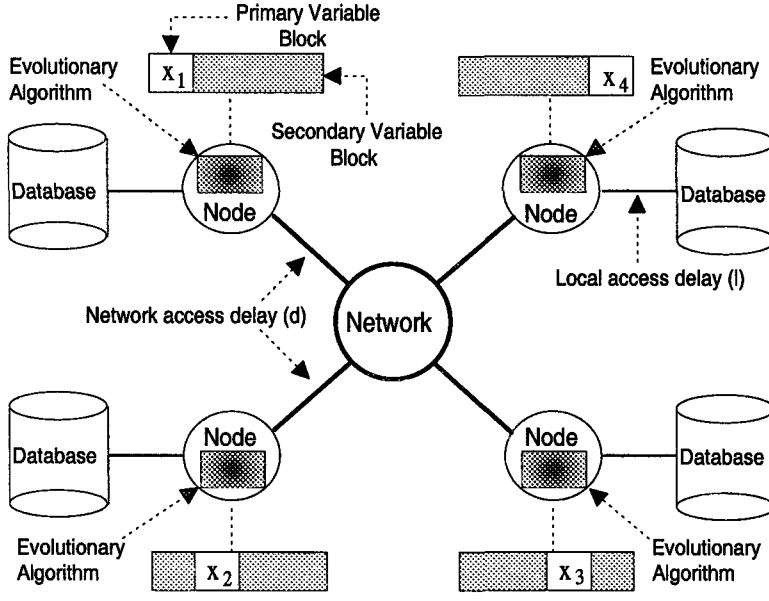


Fig. 5.1 Organization of distributed coevolutionary computation.

In this model of computation, the evolutionary algorithm at any node i performs an evolutionary search based on its primary variable block x_i using *local and rapidly* accessible information, and it is assumed that accessing the interconnection network for purposes of communication between the nodes is *delay-prone*, and so each node must perform a large number of local computations between communication cycles in interest of efficiency.

Global convergence and convergence rate results similar to those in the previous chapter are developed in Section 5.2 for this class of distributed coevolutionary algorithms. Section 5.3 presents an analysis of the relative computational delays of the centralized and distributed algorithms when they are implemented in a network environment.

This chapter is based on material that appears in [Subbu and Sanderson, 2003a], and earlier in [Subbu and Sanderson, 2000].

5.2 Theory

This section develops a general model of distributed coevolutionary algorithms wherein cooperating algorithm components are distributed over a network, and each algorithm component works primarily in a subspace of the overall problem. This approach to optimization is similar in spirit to the block Jacobi [Bertsekas and Tsitsiklis, 1997] and coordinate descent [Luenberger, 1984] methods popular in the non-linear programming community. However, the fundamental difference is that these non-linear programming techniques are deterministic and gradient based while evolutionary algorithms are not.

5.2.1 *Notation*

- Given a node i, x_i is its primary variable set, while $\overline{x}_i$ is its secondary variable set.
- $(x_i^*|\overline{x}_i) = \arg \max_{x_i \in \mathcal{X}_i} \psi(x_i|\overline{x}_i)$ is the global optimizer in the restricted space $(\cdot|\overline{x}_i)$.
- μ_{n_i} is the Lebesgue measure on $\mathbb{R}^{n_i}$ or on $\mathcal{X}_i \subset \mathbb{R}^{n_i}$.
- dx_i is compactly the infinitesimal volume element $dx_{i_1} \cdot dx_{i_2} \cdots dx_{i_{n_i}}$ in n_i dimensional real space.
- $B((x_i|\overline{x}_i), \epsilon) = \{z_i \in \mathcal{X}_i : \|x - z\| \leq \epsilon\}$, where $x = (x_i|\overline{x}_i), z = (z_i|\overline{x}_i)$, and $B(\overline{x}_i, \epsilon) = B((x_i^*|\overline{x}_i), \epsilon)$.
- $A(\overline{x}_i, \epsilon) = \{z_i \in \mathcal{X}_i : |\psi(x_i^*|\overline{x}_i) - \psi(z_i|\overline{x}_i)| \leq \epsilon\}$.

5.2.2 *Local Convergence*

The local evolutionary search in the primary subspace of each node is based on the evolutionary algorithm described in Section 4.2. In making this transition, the assumptions in Section 4.2.3 are mapped so that they are applicable to each of the subspaces $\mathcal{X}_i$.

(a) $0 < \psi(x_i|\overline{x}_i) < \infty$ for all $x_i \in \mathcal{X}_i, \overline{x}_i$.

(b) $Q_{(i,g)}(z_i, dx_i) = q_{(i,g)}(z_i, x_i)\mu_{n_i}(dx_i)$, and $\max_{z_i, x_i \in \mathcal{X}_i} q_{(i,g)}(z_i, x_i) < \infty$.

(c) The global optimizer $(x_i^*|\overline{x}_i)$ is unique, and there exists an $\epsilon > 0$ such that $\psi(\cdot)$ is continuous in the finite region $B(\overline{x}_i, \epsilon)$.

(d) There exists a $\delta > 0$ such that the sets $A(\overline{x}_i, \epsilon)$ are connected for all $\epsilon : 0 < \epsilon < \delta$.

(e) For any $z_i \in \mathcal{X}_i$ and $g \to \infty$ the sequence of probability measures $Q_{(i,g)}(z_i, dx_i)$ weakly converges to the probability measure concentrated at the point z_i.

(f) For any $\epsilon > 0$ there exists a $\delta > 0$ and a natural number k such that $P_{(i,g)}(B(\overline{x}_i, \epsilon)) \geq \delta$ for all $g \geq k$.

(g) $P_{(i,0)}(B((x_i|\overline{x}_i), \epsilon)) > 0$ for all $\epsilon > 0, x_i \in \mathcal{X}_i, \overline{x}_i$.

Based on these assumptions Results 4.1, 4.2, 4.3, 4.4 are mapped accordingly so that they hold for the evolutionary search in each subspace $\mathcal{X}_i$. Then, as $N \to \infty$ the distribution of samples in a local search would follow the sequence

$$P_{(i,g+1)}(dx_i) = \frac{\int_{\mathcal{X}_i} P_{(i,g)}(dz_i)\psi(z_i|\overline{x}_i)Q_{(i,g)}(z_i, dx_i)}{\int_{\mathcal{X}_i} P_{(i,g)}(dz_i)\psi(z_i|\overline{x}_i)} \tag{5.2}$$

and the above distribution sequence converges to the distribution concentrated at $(x_i^*|\overline{x}_i)$ as $g \to \infty$.

It is of interest to consider the convergence and convergence rate of the local computations in any subspace $\mathcal{X}_i$ for the unimodal and bimodal objectives based on Gaussians. For this, the fact that the Gaussian (resp. linear sum of Gaussians) objective in any subspace $\mathcal{X}_i$ given variable assignments in the other $p-1$ subspaces is still a Gaussian (resp. linear sum of Gaussians) presents an advantage. If the conditions for the evolutionary algorithm that performs local search in any subspace $\mathcal{X}_i$ are exactly the conditions for the algorithm (discussed in Section 4.3) that performs search in space $\mathcal{X}$, then Theorems 4.1, 4.2, 4.3, 4.4 would hold without modifications except that space $\mathcal{X}$ is to be interpreted as subspace $\mathcal{X}_i$, and the local population distribution at node i would *converge geometrically* to $(x_i^*|\overline{x}_i)$. In the event that $(x_i^*|\overline{x}_i)$ is not unique, the theoretical population distribution would converge to multiple points. In practice, it may be assumed without loss of generality that the distribution converges to only one such point.

The local searches described above are initialized with a randomly selected consistent vector of variables x_g. This vector is broadcast to all

nodes. The local search at node i starting from this point may be represented by a mapping $T_i : \mathcal{X} \longrightarrow \mathcal{X}_i$ that generates the sequence

$$x_{(i,g+m+1)} = T_i(x_{(1,g)}, \cdots, x_{(i-1,g)}, x_{(i,g+m)}, x_{(i+1,g)}, \cdots, x_{(p,g)}) \quad m \geq 0$$

Then, as m increases

$$x_g^{(i)} = (x_{(1,g)}, \cdots, x_{(i-1,g)}, x_{(i,g+m)}, x_{(i+1,g)}, \cdots, x_{(p,g)})$$

and $x_g^{(i)}$ would converge geometrically to $(x_i^*|\overline{x}_i)$, where $x_g^{(i)}$ is the result of m generations of local search at node i, starting from point x_g. From a practical perspective, it is convenient to select $x_g^{(i)}$ as the best vector (after m generations of local search) in the population at node i.

Given that the process of local convergence at a node for Gaussian-based objectives is understood, the focus is now on distributed global convergence to the point x^*, which is the essentially the overall goal. This is discussed next.

5.2.3 *Global Convergence*

Presented first are communication policies that would enable distributed global convergence for a unimodal Gaussian objective and for a bimodal objective based on a linear combination of Gaussians, whose optima are aligned favorably with respect to the search directions. Presented next are communication policies that would enable distributed global convergence for arbitrarily aligned bimodal objectives.

5.2.3.1 *Convergence for a Unimodal Objective*

Let $\mathcal{Z}_g = \left\{x_g, x_g^{(1)}, \cdots, x_g^{(p)}\right\}$, where $x_g^{(i)}$ is the *best vector* (after m generations of local search) in the population at node i and let $S : \mathcal{X} \longrightarrow \mathcal{X}$ represent the computation that selects that vector from $\mathcal{Z}_g - x_g$ which has the highest fitness and makes it the new iterate x_{g+1} only if its fitness is greater than that of x_g (else $x_{g+1} = x_g$). The computation $x_{g+1} = S(x_g)$ represents a *global iteration* that encapsulates the combined m-step local search at each node and the intercommunication operation that facilitates selection and update of new iterates. It can be easily seen that the mapping S generates a non-decreasing sequence $\{\psi(x_g)\}$, and the theorem below describes the rate of global convergence for a unimodal objective.

Theorem 5.1 *Let $\psi(x) = \mathsf{N}[x^*, K]$ with positive definite K. Then, the sequence $\{\psi(x_g)\}$ generated by the mapping S converges geometrically to $\psi(x^*)$.*

Proof: Let x_g be given. The objective function in each subspace $\mathcal{X}_i$ is still a Gaussian with a unique optimum. If x_g does not correspond to the optimum in $\mathcal{X}_i$, a local search must yield a point whose fitness is greater than that of x_g. If x_g does correspond to the optimum in $\mathcal{X}_i$, a local search does not change position. Therefore, unless x_g is an optimum in $\mathcal{X}$, at least one point in the set $\mathcal{Z}_g - x_g$ will have fitness greater than that of x_g, and $\psi(x_{g+1}) > \psi(x_g)$; which implies that $|\psi(x_{g+1}) - \psi(x^*)| \leq \alpha|\psi(x_g) - \psi(x^*)|, \alpha \in [0, 1)$. Since $\psi(x)$ is unimodal with a unique optimum, this shows that the iteration sequence $\{\psi(x_g)\}$ converges geometrically to $\psi(x^*)$. ■

The next theorem describes the rate of global convergence of the distributed algorithm applied to a bimodal objective function whose optima are aligned favorably with respect to the search coordinates.

5.2.3.2 *Convergence for a Coordinate Aligned Bimodal Objective*

Theorem 5.2 *Let $\psi(x) = \omega_a \mathsf{N}[x_a^*, K_a] + \omega_b \mathsf{N}[x_b^*, K_b]$ with positive definite K_a, K_b, and $\omega_a, \omega_b > 0$. Also, let x_a^* be the global optimizer, and let x_a^*, x_b^* be such that*

$$x_a^* = (x_1^*, \cdots, x_{i-1}^*, x_i^*, x_{i+1}^*, \cdots, x_p^*)$$
$$x_b^* = (x_1^*, \cdots, x_{i-1}^*, \tilde{x}_i^*, x_{i+1}^*, \cdots, x_p^*)$$

where x_a^*, x_b^* differ only in the coordinates in the subspace $\mathcal{X}_i$. Then, the rate of convergence to the global optimizer x_a^* is similar to that in Theorem 5.1.

Proof: Let x_g be given. The objective function in each subspace $\mathcal{X}_i$ is still a linear combination of Gaussians. However, the optima in $\mathcal{X}_i$ need not be unique. Based on an earlier assumption, a local search would still converge geometrically to one of the two optima in $\mathcal{X}_i$. If x_g does not correspond to an optimum in $\mathcal{X}_i$, a local search must yield a point whose fitness is greater than that of x_g. If x_g does correspond to an optimum in the subspace $\mathcal{X}_i$ and the optimum in $\mathcal{X}_i$ is not unique, it is satisfactory to make the mild assumption that a local search does not change position. So, a local search must either yield a point whose fitness is greater than that of x_g, or it does not change position. Therefore, unless x_g is an optimum in $\mathcal{X}$, at least one point in the set $\mathcal{Z}_g - x_g$ will have fitness greater than

that of x_g, and $\psi(x_{g+1}) > \psi(x_g)$; which implies that $|\psi(x_{g+1}) - \psi(x_a^*)| \leq \alpha|\psi(x_g) - \psi(x_a^*)|, \alpha \in [0, 1)$. However, if x_g is an optimum in $\mathcal{X}$, it is either x_a^* or it is x_b^* and x_a^* can be found using purely local search in $\mathcal{X}_i$. Therefore, the rate of convergence of the search is similar to that in Theorem 5.1. ■

The above result shows that there exists some favorable transformation of coordinates such that the complexity of the distributed search over a bimodal function may be reduced to that of a search over a unimodal function, simplifying the search task. In general instances, when a favorable transformation of coordinates (or additional problem specific information) is unavailable, randomization is introduced in the selection of new iterates. Such a class of methods is described next.

5.2.3.3 *Convergence for an Arbitrarily Aligned Multimodal Objective*

Three basic stochastic coordination policies are now described and analyzed. These coordination policies facilitate global convergence for arbitrarily aligned bimodal objective functions.[1] Consider a bimodal objective where x_a^* is the global optimizer and x_b^* is the other optimum such that $\psi(x_a^*) > \psi(x_b^*)$.[2] Let $C = \{z \in \mathcal{X} : \psi(z) > \psi(x_b^*)\}$. Therefore, C is the region in the space $\mathcal{X}$ that has objective values greater than those in the union of the supports of all other modes. Let $C_i \subset \mathcal{X}_i$ be the projection of the region C on each subspace $\mathcal{X}_i$ (as an example, consider the two dimensional region $\mathcal{X}$ shown in Figure 5.2, where $\mathcal{X} = \mathcal{X}_1 \times \mathcal{X}_2$.). Let $\hat{C}_i = C_1 \times \cdots \times C_{i-1} \times \mathcal{X}_i \times C_{i+1} \times \cdots \times C_p$, and let $M = \hat{C}_1 \cup \cdots \cup \hat{C}_p$.

Coordination Policy–A

Let $x_g{}'$ be a randomly generated point that is the result of a joint computation by the nodes. Let $S : \mathcal{X} \longrightarrow \mathcal{X}$ with a small probability γ set $x_g{}'$ as the new iterate x_{g+1}, and with probability $(1 - \gamma)$ use the deterministic scheme (selection from the set $\mathcal{Z}_g$). For this it is assumed that there is a memory that remembers previously attained points, so that those points are not lost. Also, it is assumed that as long as the deterministic selection scheme makes sufficient progress, randomization is avoided by temporarily setting $(\gamma = 0)$. This latter assumption allows the distributed search to deterministically climb a promising peak.

[1] These results are also applicable to multimodal objectives based on Gaussians.

[2] For multimodal objectives x_b^* would be the point corresponding to the second highest mode.

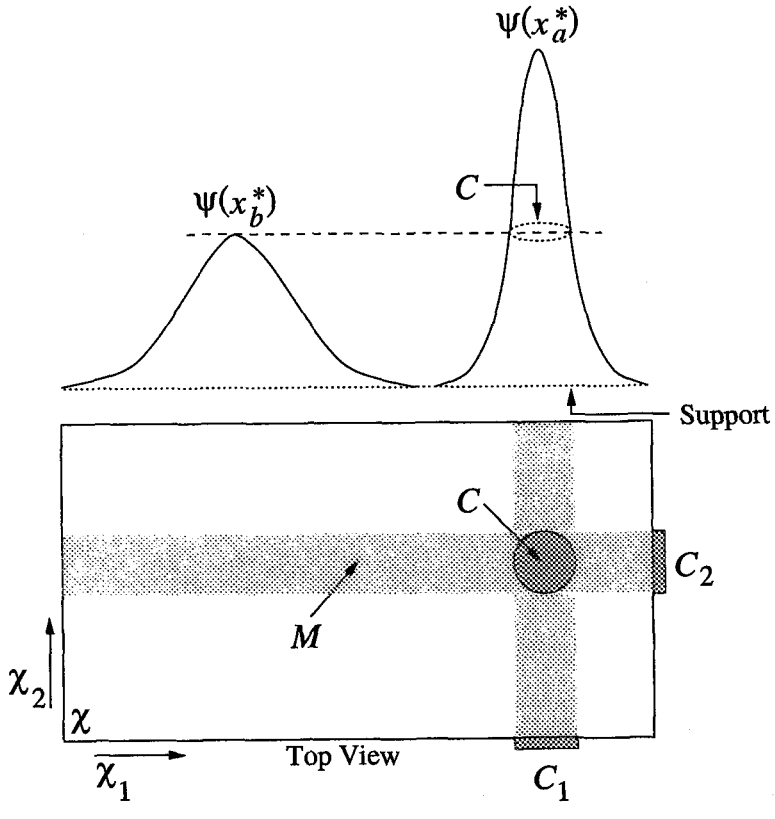


Fig. 5.2 Example two-dimensional space that shows the regions C and M.

Once an iterate enters the region C, the above coordination policy would ensure that the global optimizer is attained deterministically. In the context of the distributed search there exists the larger region M, $C \subset M$ such that if $x_g{}'$ falls in this region and is selected as the new iterate x_{g+1}, then a local search by at least one node would find a point in C. Then, the probability that $x_g{}'$ falls in the region M is given by $\mu_n(M)/\mu_n(\mathcal{X})$. The probability that $x_g{}'$ falls in the region M and is selected as the new iterate is therefore $\gamma\mu_n(M)/\mu_n(\mathcal{X})$. This is also the probability that a randomly selected point $x_g{}'$ would lead to global convergence under this coordination policy.

Coordination Policy–B

Let $x_g{}'$ be a randomly generated point that is the result of a joint computation by the nodes. Let $\mathcal{Z}_g{}' = \left\{x_g, x_g^{(1)}, \cdots, x_g^{(p)}, x_g{}'\right\}$, and let $S : \mathcal{X} \longrightarrow \mathcal{X}$ represent the computation that selects that vector from $\mathcal{Z}_g{}' - x_g$ which has the highest fitness and makes it the new iterate x_{g+1} only if its fitness is greater than that of x_g (else $x_{g+1} = x_g$).

Once a randomly generated point $x_g{}'$ enters the region C, and even if no other points from the set $\mathcal{Z}_g{}'$ are in this region, the above coordination policy would ensure that $x_g{}'$ becomes the new iterate. From this stage on the global optimizer will be attained deterministically without any stochastic assistance. Therefore, the probability that a randomly selected point $x_g{}'$ would lead to global convergence under this coordination policy, is given by $\mu_n(C)/\mu_n(\mathcal{X})$.

Coordination Policy–C

Let $\{x_g{}'\}$ be a set of randomly generated points whose elements are results of joint computations by the nodes. Let $\mathcal{Z}_g{}'' = \left\{x_g, x_g^{(1)}, \cdots, x_g^{(p)}, \{x_g{}'\}\right\}$, and let $S : \mathcal{X} \longrightarrow \mathcal{X}$ represent the computation that selects that vector from $\mathcal{Z}_g{}'' - x_g$ which has the highest fitness and makes it the new iterate x_{g+1} only if its fitness is greater than that of x_g (else $x_{g+1} = x_g$).

Once a point from the set $\{x_g{}'\}$ enters the region C, and even if no other points from the set $\mathcal{Z}_g{}''$ are in this region, the above coordination policy ensures that point becomes the new iterate. From this stage on the global optimizer will be attained deterministically without any stochastic assistance. The probability that a random selection $x_g{}'$ would lead to global convergence under this coordination policy, is computed below.

Assume that each point in the set $\{x_g{}'\}$ is generated independently, and that this set has cardinality c. Then, the probability that at least one point from this set falls in the region C is given by

$$1 - \left(1 - \frac{\mu_n(C)}{\mu_n(\mathcal{X})}\right)^c \approx 1 - \exp\left(-c\frac{\mu_n(C)}{\mu_n(\mathcal{X})}\right) \quad \text{assuming } \mu_n(C)/\mu_n(\mathcal{X}) \ll 1$$

Discussion

A direct comparison of "Coordination Policy–B" and "Coordination Policy–C" reveals that the latter policy has a higher chance of global convergence

when $c > 1$. From a theoretical perspective, "Coordination Policy–A" appears to have the potential to do better than "Coordination Policy–B" since the space C is subsumed by the space M. However, from a practical perspective[3], it is impossible to assert that if $x_g{}'$ falls in the region $M - C$ and is selected as the new iterate, then with probability 1 a local search by at least one node would find a point in C within some fixed number of generations. So, for "Coordination Policy–A" the probability $\gamma\mu_n(M)/\mu_n(\mathcal{X})$ serves as the upper bound, and the lower bound is $\gamma\mu_n(C)/\mu_n(\mathcal{X})$. Therefore, the lower bound probability associated with "Coordination Policy–A" is less than or equal to the probability associated with "Coordination Policy–B," and by a transitive argument it is less than the probability associated with "Coordination Policy–C" when $c > 1$.

The important insight on these coordination policies is that for an arbitrarily multimodal objective, there is only the guarantee that such policies will enable global convergence as time tends to infinity. It is not possible to predict their convergence rates, unless of course these objectives are favorably transformed as discussed in Theorem 5.2.

5.3 Computational Delay Analysis

This section analyzes computational delays associated with the centralized and distributed algorithms when they are implemented in a network environment. Computational delays are modeled as functions of several variables including network delays, local database access delays, number of nodes, and frequency of intercommunication. Each of the p nodes in the network environment is assumed to have a locally resident database (see Figure 5.1), information from which is necessary for function evaluations of the form $\psi(x)$ in (5.1). d is the network access delay for accessing a node from any other node. l is the local database access delay for retrieving information corresponding to any variable block x_i from the local database at node i. In these models, the computational cost of applying the selection and stochastic variation operations on a local population to create new individuals is assumed to be negligible compared to the computational cost of evaluating new individuals, which requires both local and network accesses.

[3]Given a finite population size.

5.3.1 *Centralized Computation*

In this model of computation, only one of the p nodes performs the evolutionary search while the other $p-1$ nodes provide information on request. Evaluation of an individual in the population requires a local information request and $p-1$ network information requests. m_c is the problem dependent population size. Then, the computational delay per generation of evolution, t_c, is given by $t_c = t_{\text{local}} + t_{\text{comm}}$, where $t_{\text{local}} = lm_c$, and $t_{\text{comm}} = (2d+l)(p-1)m_c$. The term $2d$ in t_{comm} models a network access cycle.

5.3.2 *Distributed Coevolutionary Computation*

In this model of computation, each of the p nodes participates in a coevolutionary search, and it is assumed that the computation at each node proceeds synchronously. It is assumed that there exists a coordination mechanism common to the p nodes (resident at some node), and it takes a network access of each of the other nodes to collect partial results, a network access of each these nodes to evaluate the result of any joint computation in the set $\{x_g{}'\}$, and a network access of each of these nodes for transmitting updates. f is the problem dependent normalized frequency of coordination–a number in the range $(0,1]$. When $f=1$ a coordination occurs every generation of local search, and as $f \to 0$ the frequency of coordination drops. m_d is the problem dependent local population size at each node. It is assumed that $m_d \leq m_c$, since a local search at a node is over a space smaller than the space in the centralized search. Then, the computation delay per generation of evolution, t_d, is given by $t_d = t_{\text{local}} + t_{\text{comm}}$, where $t_{\text{local}} = lm_d$, and $t_{\text{comm}} = [4d + (2d+l)c](p-1)f$. The term $4d$ in t_{comm} models two network access cycles required to collect and transmit results, and the term $(2d+l)c$ models the cost of accessing a node for evaluating c joint computations.

5.3.3 *Computational Advantage*

The nature of the ratio $\frac{t_c}{t_d}$, which is a function of the variables l, d, p, f, c, and the population sizes m_c, m_d, is of interest. This ratio represents the *computational advantage* (a large value of the ratio corresponds to a large advantage) of the distributed coevolutionary computation and holds under the idealized assumption that the centralized and distributed algorithms

are capable of producing similar results in the same number of generations. It is of interest to study the relationship of this ratio to the following factors:

- Frequency of coordination f.
- Number of computational nodes p.
- Population sizes m_c, m_d of the algorithms.
- Limiting cases of the ratio of local access and network access delays $\frac{l}{d}$. When $\frac{l}{d} \ll 1$ each node can access its local database much faster than it can access another node, and when $\frac{l}{d} > 1$ a network access is faster that a local database access.
- Cardinality c of the set $\{x_g'\}$.

The ratio $\frac{t_c}{t_d}$ is written as $\frac{t_c}{d}\frac{d}{t_d}$, where

$$\frac{t_c}{d} = \frac{l}{d}m_c + \left(2 + \frac{l}{d}\right)(p-1)m_c \tag{5.3}$$

and

$$\frac{t_d}{d} = \frac{l}{d}m_d + \left[4 + \left(2 + \frac{l}{d}\right)c\right](p-1)f \tag{5.4}$$

Figures 5.3, and 5.4 show the nature of the trade-offs between the computational advantage and frequency of coordination for several values of the ratio $\frac{l}{d}$ and number of nodes p, respectively when $c = 1$ and $c = 10$. For these trade-offs, a constant population size ($m_c = m_d = 100$) is used for the centralized and distributed algorithms. These trade-offs show that in general a higher computational advantage is achievable for a distributed algorithm implementation that can tolerate infrequent coordination. Even if a coordination operation were to occur every generation of evolution, there is a significant advantage to distributed coevolutionary computation provided the ratio $\frac{l}{d}$ is sufficiently smaller than 1.

When $\frac{l}{d} \ll 1$, a distributed algorithm is highly scalable with respect to the number of computational nodes–the trade-off curves have a tighter spread over a wider range of number of computational nodes. Even when the local access delay dominates the network access delay (for example $\frac{l}{d} = 100$), there is a significant advantage to distributing the computation when the number of nodes is large. Figure 5.4 shows that when

the cardinality of the set $\{x_g'\}$ increases (resulting in a higher number of network-based evaluations during every coordination operation), the advantage to distributing the implementation decreases. However, from a practical perspective, a larger cardinality c may lead to faster convergence of the distributed algorithm thereby potentially offsetting any performance degradations.

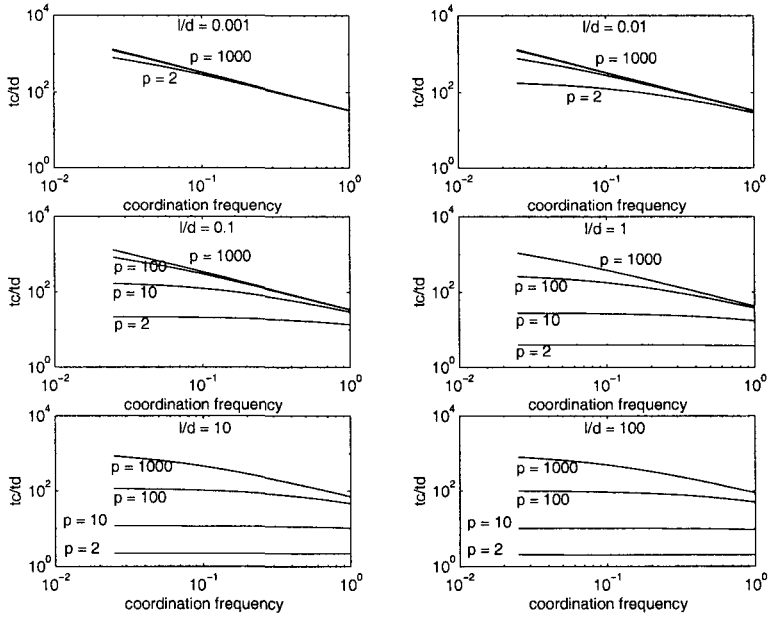


Fig. 5.3 Nature of the ratio $\frac{t_c}{t_d}$ as a function of $\frac{l}{d}, p, f$ when $m_c = m_d = 100, c = 1$.

Bounds for the computational advantage $\frac{t_c}{t_d}$ are now derived under the limiting conditions $\frac{l}{d} \to 0$ and $\frac{l}{d} \to \infty$. First, let $\frac{l}{d} \to 0$. Then, (5.3) may be approximated by

$$\frac{t_c}{d} \approx 2(p-1)m_c \tag{5.5}$$

and (5.4) may be approximated by

$$\frac{t_d}{d} \approx (4+2c)(p-1)f \tag{5.6}$$

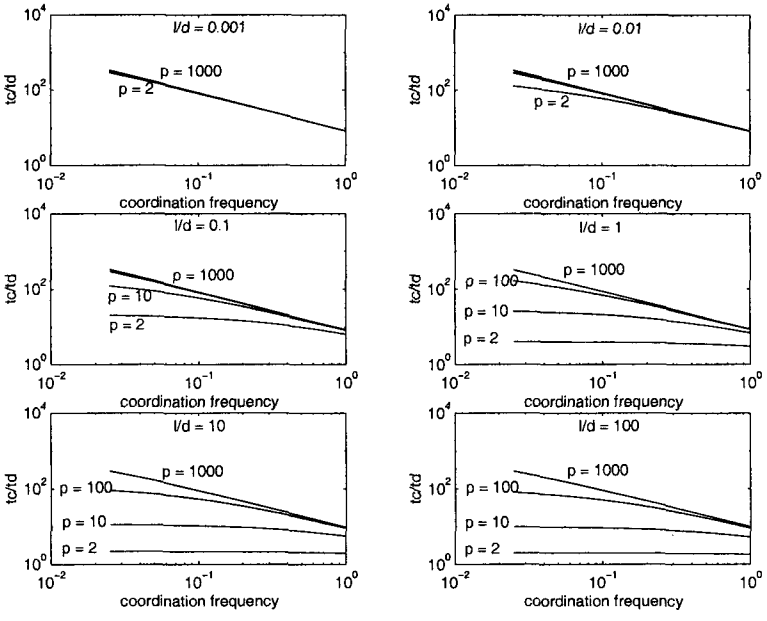


Fig. 5.4 Nature of the ratio $\frac{t_c}{t_d}$ as a function of $\frac{l}{d}, p, f$ when $m_c = m_d = 100, c = 10$.

Therefore, the limit of the ratio of (5.3) and (5.4) when $\frac{l}{d} \to 0$, may be computed using (5.5) and (5.6), and is given by

$$\lim_{\frac{l}{d} \to 0} \frac{t_c}{t_d} = \frac{m_c}{(c+2)f} \tag{5.7}$$

Similarly, when $\frac{l}{d} \to \infty$, (5.3) may be approximated by

$$\frac{t_c}{d} \approx \frac{l}{d} m_c + \frac{l}{d}(p-1)m_c = \frac{l}{d} p m_c \tag{5.8}$$

and (5.4) may be approximated by

$$\frac{t_d}{d} \approx \frac{l}{d} m_d + \frac{l}{d} c(p-1)f \tag{5.9}$$

Therefore, the limit of the ratio of (5.3) and (5.4) when $\frac{l}{d} \to \infty$, may be computed using (5.8) and (5.9), and is given by

$$\lim_{\frac{l}{d} \to \infty} \frac{t_c}{t_d} = \frac{pm_c}{m_d + c(p-1)f} \tag{5.10}$$

This page is intentionally left blank

Chapter 6

Performance Evaluation Based on Ideal Objectives

6.1 Introduction

The two previous chapters respectively focused on developing a theoretical foundation for centralized evolutionary algorithms and for distributed coevolutionary algorithms. This theoretical foundation describes evolutionary and coevolutionary algorithms in terms of creation and evolution of sampling distributions over the feasible space. Using this approach, global convergence and convergence rate results are derived for certain basic classes of objective functions.

This chapter presents simulation results based on several ideal objectives function classes. The first test function class is a unimodal Gaussian. The second test function class is a bimodal Gaussian, and for functions in this class, their optima are aligned according to the description in Theorem 5.2. The third test function class is also a bimodal Gaussian, but corresponds to a general case objective with no favorable coordinate transformation. The fourth test function class is a set of increasingly difficult planar tile layout problems. Finally, an evolutionary algorithm is developed and evaluated for the general design-supplier-manufacturing decision problem.

This chapter is based in part on material that appears in [Subbu and Sanderson, 2003a].

6.2 Gaussian Objectives

This section presents simulation results using three test function classes based on Gaussians and a linear combination of Gaussians. The first test function class is unimodal in nature. The second test function class is bimodal in nature, and for functions in this class, their optima are aligned

according to the description in Theorem 5.2. The third test function class is also bimodal in nature, but corresponds to a general case objective with no favorable coordinate transformation.

Twenty stochastically generated test functions from each of the three classes are used for the evaluation, which is based on a total of 60 test functions. Each test function consists of four variables, and height contours of the corresponding sample two-variable versions are shown in Figure 6.1.

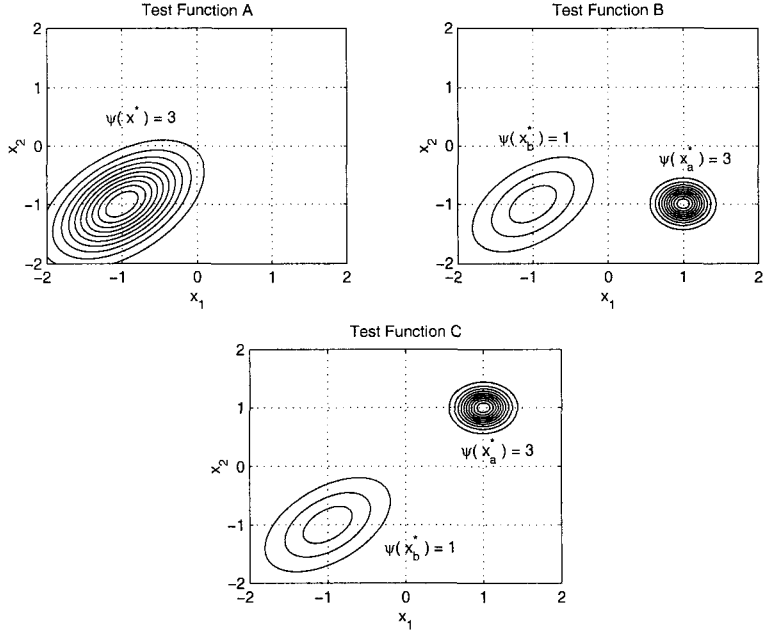


Fig. 6.1 Two-variable example functions from each of the ideal test function classes.

Function-A is the unimodal function class

$$\psi(x) = 3 \exp\left(-\frac{1}{2}(x - x^*)^T K^{-1}(x - x^*)\right) \tag{6.1}$$

where $x^* = [-1, -1, -1, -1]^T$, and twenty instances of the covariance matrix K are randomly constructed using uniform standard deviations in the range $[0.5, 1.0]$, and uniform correlation coefficients in the range $[-0.6, 0.6]$. Each randomly generated K is used to construct a test function in this class.

Function-B is the bimodal function class

$$\psi(x) = 3 \exp\left(-\frac{1}{2}(x - x_a^*)^T {K_a}^{-1}(x - x_a^*)\right) + \exp\left(-\frac{1}{2}(x - x_b^*)^T {K_b}^{-1}(x - x_b^*)\right) \tag{6.2}$$

where $x_a^* = [1, 1, -1, -1]^T$, $x_b^* = [-1, -1, -1, -1]^T$, twenty instances of the covariance matrix K_a are randomly constructed using uniform standard deviations in the range $[0.2, 0.5]$, and uniform correlation coefficients in the range $[-0.6, 0.6]$, and twenty instances of the covariance matrix K_b are randomly constructed using uniform standard deviations in the range $[0.5, 1.0]$, and uniform correlation coefficients in the range $[-0.6, 0.6]$. Each random combination of K_a and K_b is used to construct a test function in this class.

Function-C is also a bimodal function class, and is similar to test functions in the Function-B class except that $x_a^* = [1, 1, 1, 1]^T$. Gaussian-based test functions are used since they are representative of the basic function classes of interest and offer consistency with the theory presented in this book.

The centralized and distributed algorithms are implemented in a two-node distributed environment. For the case of the distributed algorithm, the first two variables form the primary variable set for the first node while the last two variables form the primary variable set for the second node. The representation is a real encoded four dimensional vector of variables. Gaussian mutation with parent-child competition introduces stochastic variation in the samples. The range of each variable is $[-2, 2]$, consistent with the description in Figure 6.1. The simulation parameters are: Network delay $d = 10$, Local access delay $l = 1$, Number of nodes $p = 2$, Coordination frequency (for the distributed algorithm) $f = 0.1$, Population sizes: $m_c = 200, m_d = 50$. The distributed communication scheme follows the discussion in Coordination Policy–A (Section 5.2.3.3).

Figure 6.2 shows the average (and standard deviation) convergence performance over 400 trials (20 trials for each of the 20 randomly generated test functions in each of the three classes) of the evolution of the mean fitness of the population distribution of the centralized algorithm, and the evolution of the fitness of the global iterates of the distributed algorithm, considered with respect to a system that geometrically seeks with contraction coefficient $\alpha = 0.98$, a value of 3 starting from 0. The abscissa in these performance plots corresponds to search generations. The above results

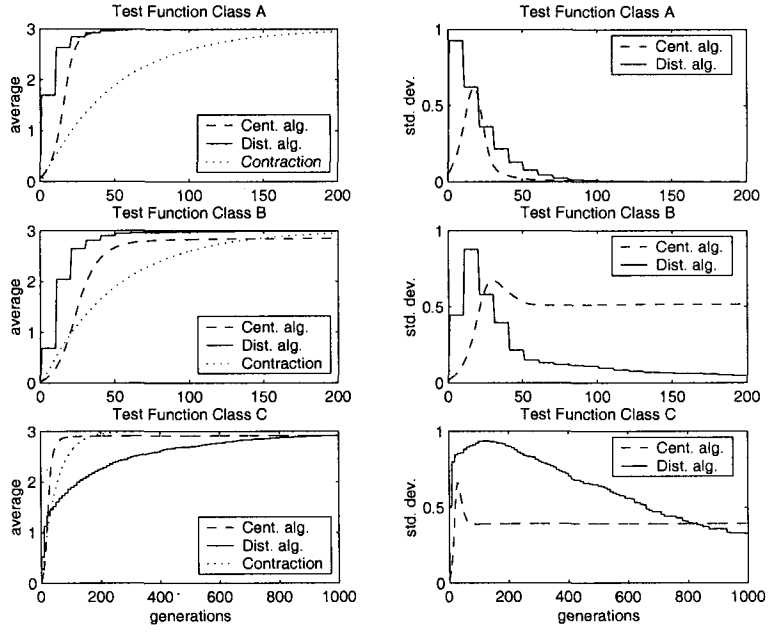


Fig. 6.2 Convergence performance over 400 trials of the centralized and distributed algorithms applied to test functions from each of the three ideal test function classes, and considered with respect to a system that geometrically seeks with contraction coefficient $\alpha = 0.98$, a value of 3 starting from 0.

show that for all the considered test functions the centralized algorithm on average clearly converges geometrically as predicted earlier, and the distributed algorithm on average also clearly converges geometrically for the unimodal and coordinate aligned bimodal objectives as predicted earlier. For arbitrarily aligned bimodal objectives in function class C, the convergence rate of the distributed algorithm is weaker than geometric, and is highly dependent on the coordination scheme incorporated, as stated earlier. Occasionally, for certain randomly generated test functions in the classes B and C, both the centralized and distributed algorithms converge sub-optimally, resulting in nonzero standard deviations at higher generation counts. Such behavior is typical in all randomized algorithms including evolutionary algorithms.

Figure 6.3 shows the relative time performance of the centralized and distributed algorithms from Figure 6.2 when network and local access delays are considered. In spite of the fact that function-C is more difficult

that the others for the distributed algorithm, its convergence behavior is consistently better than that of the centralized algorithm when network delays are considered.

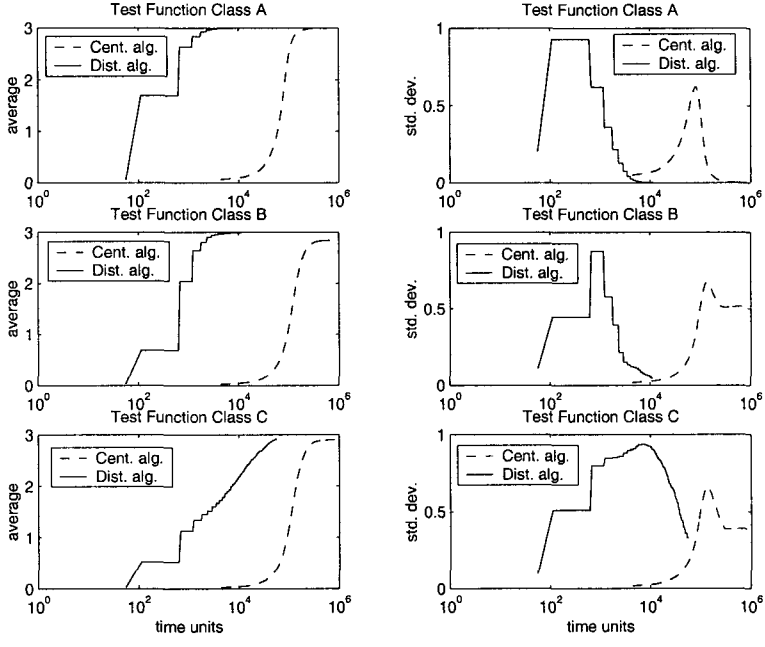


Fig. 6.3 Time performance over 400 trials of the centralized and distributed algorithms applied to test functions from each of the three ideal test function classes.

6.3 Planar Tile Layout Problems

In this section, the centralized and distributed algorithms are evaluated by their ability to solve select planar tile layout problems. The problem is an arrangement of tiles to cover a unit square area within a larger bounded region such that the tiles do not mutually overlap (see Figure 6.4). The tiles are arranged using translations in the two-dimensional space; tile rotations are not allowed. If the tiles were to mutually overlap the arrangement would incur a penalty directly proportional to the area of overlap. Though the tiles may not exceed the layout bound, they may overlap the shaded region, but then such a layout would incur a penalty directly proportional

to the area of overlap.

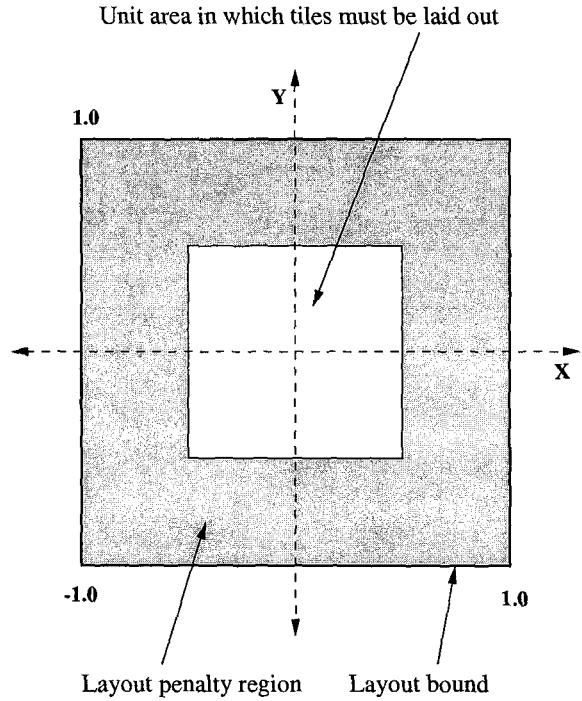


Fig. 6.4 Tile layout problem.

In the centralized algorithm implementation, the positions of all the tiles can be changed simultaneously, while in the distributed algorithm implementation each node controls the position of a unique tile, and it is assumed that internode communications are delay-prone. The representation used is a concatenation of the real encoded Cartesian coordinates of an anchor point of each tile, and each search variable has a range of $[-1, 1]$. Each complete tile-set arrangement is scored directly proportional to the percentage of area covered; the objective being its maximization.

Figure 6.5 shows three tile layouts (A, B, C) of respectively increasing search difficulty, which serve as test cases. For each of the test cases the maximum cover is achieved when the tiles do not overlap and fit exactly within the unit area shown in Figure 6.4. For layout-A an exact fit corresponding to the maximum cover has two alternatives (tile positions can be swapped), while for layouts B and C the maximum cover is achieved

only for unique respective tile-set configurations. Due to the nature of the

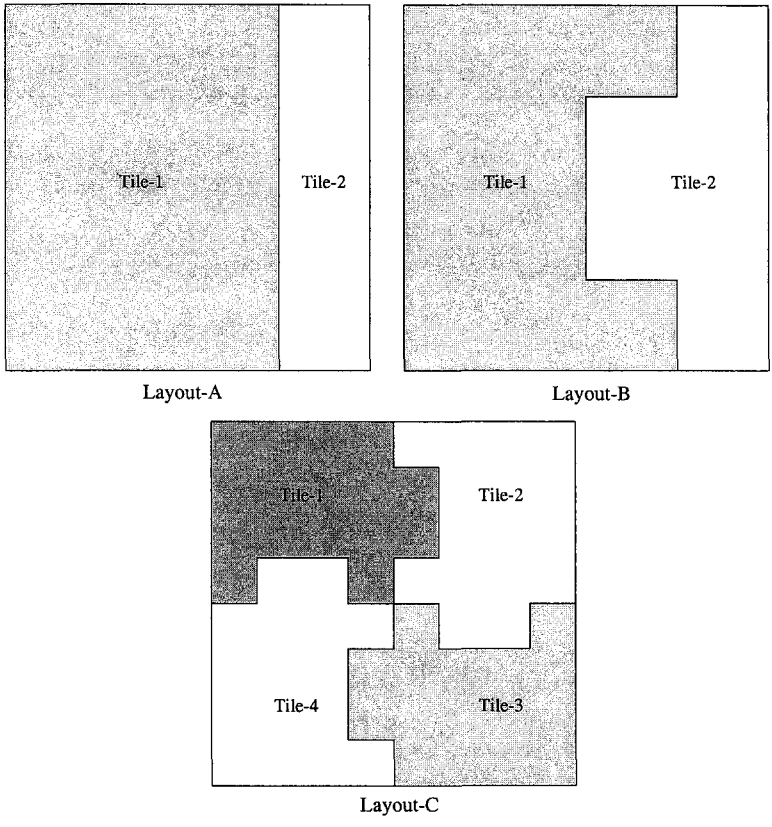


Fig. 6.5 Exact tile layouts for the three test cases.

representation used, layouts A and B correspond to four-variable search problems, while layout-C corresponds to an eight-variable search problem. The search space corresponding to layout-A has optima aligned with the search coordinates, so for this problem the distributed algorithm does not require any randomization in the selection of new iterates. However, for distributed searches corresponding to layouts B and C many competing and unfavorably aligned optima occur requiring randomization in the selection of iterates.

The simulation parameters for the respective searches corresponding to layouts A and B are: Network delay $d = 10$, Local access delay $l = 1$,

Number of nodes $p = 2$, Coordination frequency (for the distributed algorithm) $f = 0.1$, and Population sizes: $m_c = 200, m_d = 50$. The simulation parameters for the search corresponding to layout-C are: Network delay $d = 10$, Local access delay $l = 1$, Number of nodes $p = 4$, Coordination frequency (for the distributed algorithm) $f = 0.1$, and Population sizes: $m_c = 4000, m_d = 100$. Gaussian mutation with parent-child competition introduces stochastic variation in the samples. A large population size is used for the centralized search corresponding to tile layout-C primarily because of the presence of many competing local optima, and in the absence of problem specific heuristics, a large population size is a means for maintaining diversity as the search matures. The distributed algorithm on the other hand does not require large population sizes because randomization in selection of global iterates introduces the necessary search diversity. Figure 6.6 compares the average (and standard deviation) performance over 20 trials of the evolution of the mean of the population distribution of the centralized algorithm, and the evolution of the global iterates of the distributed algorithm with respect to search generations. Figure 6.7 shows the same comparison with respect to time requirements of the respective algorithms. The distributed search corresponding to this problem consistently achieves the global optimum, albeit in more generations (for the harder tile layouts–B and C), due to randomization in the selection of global iterates. These performance figures reinforce the observation that the distributed computation has a better time-performance advantage when network communication delays are considered.

6.3.1 *Discussion*

In each of the three tile layout problems, a node controls the position of a unique tile, and relies on information from the other nodes for computing the fitness of each of its alternatives. The fitness of any alternative at a given node is a conditional fitness, in other words, local fitness computations are conditioned upon the locations of representative tiles from each of the other nodes. Alternatively, the nodes by design have to cooperate to solve the global problem. One might argue that if the global fitness space is, for purposes of illustration, similar to a quadratic with no correlation between search variables, or sets of search variables, and each node controls one such set, then it would be possible to solve the problem with a one-step communication following local search at each of the nodes. When there is any significant correlation among the partitioned search variables,

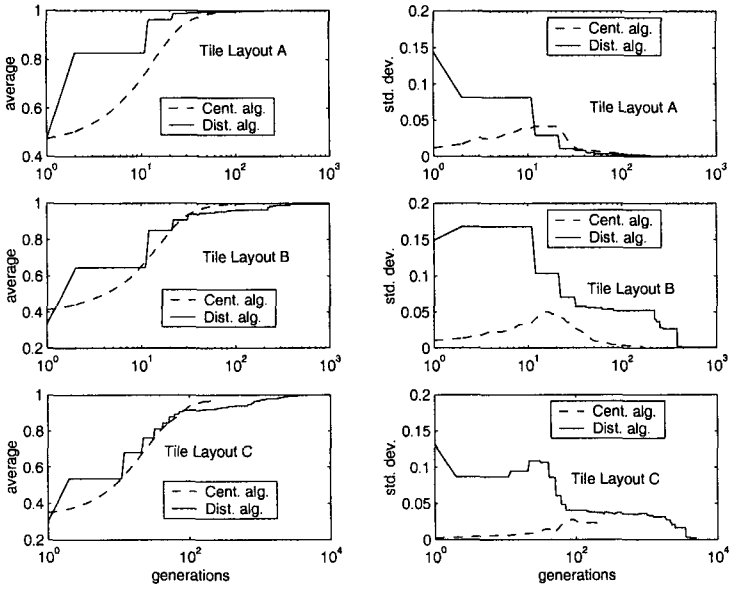


Fig. 6.6 Convergence performance over 20 trials of the centralized and distributed algorithms applied to the tile layout test cases.

communication is integral to the problem solving.

A selection of the three tile geometries is conditioned upon such an observation, and the three problems A, B, and C correspond to increasingly coupled (epistasis) problems due to the introduction of notches in their tile geometries. Tile layout-A with two rectangular tiles has lower epistasis than the other problems. In this problem, each of the two tiles can be fixed a priori at one of two extreme positions, and the other free tile may move independently to the globally optimum position. However, in any other non-extreme position, the two tiles have to move in unison (via communication) to seek the global optimum (optima in this case). Tile layout-B is a further complication due to the introduction of a significant notch, and unless each tile is placed exactly in its respective extreme position, the other tile cannot seek the global optimum without communication. Tile layout-C is clearly the most complicated with a puzzle nature to it, and the only way for a tile to seek the global optimum on its own is if all other tiles are positioned a priori in their respective extreme positions. Therefore, for each of the three tile layout problems, a low epistasis situation will arise only very remotely based on the a priori starting positions of the tiles. When

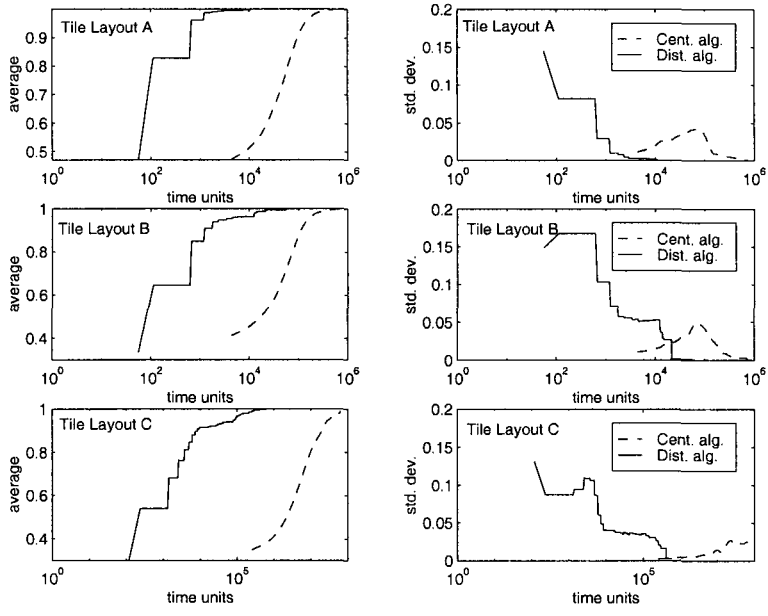


Fig. 6.7 Time performance over 20 trials of the centralized and distributed algorithms applied to the tile layout test cases.

the tile positions are randomly initialized, as in this presentation, there will be multiple optima encountered (by construction of the problems), and low epistasis is not a concern.

Further, the experimental investigation reveals that especially for Tile layouts B and C, the distributed algorithm does not reliably converge to the global optimum without explicit randomization in the communication, showing that these are not trivial problems for a distributed algorithm. In other words, unless there is randomization introduced in the selection of a global state, there is a good chance of convergence to a local optimum via distributed local hill climbing.

6.4 Design-Supplier-Manufacturing Problem

This section develops evolutionary and coevolutionary algorithms for the general design-supplier-manufacturing decision problem described in Section 3.2, and evaluates it on example problems that have the same assign-

ment constraints and structure as the general problem but whose objectives are simpler than those of the general problem. Simplifying the objectives allows creation of test problems with well known global optima without compromising the structure of the general problem.[1] Also, from the perspective of design of an evolutionary algorithm and operators, the problem's assignment constraints and structure are principal factors.

6.4.1 *Representation*

To facilitate a description of the representation for evolutionary optimization, it is convenient to use the example structure shown in Figure 3.1 as a starting point. Figure 6.8 shows a two dimensional matrix that facilitates the generation and evaluation of potential assignments in Figure 3.1. In this matrix, each slot is a gene that can assume a real value in the range $[0, 1]$. Each column in this matrix corresponds to a unique edge from the bipartite graph between parts and their supply sources. The genes in the topmost row of the matrix are weights signifying relative preferences of combined selection of parts and their respective supply sources, and each gene may have a coupling with one or more of its neighboring genes. Consider the set of n supplier–part edges for parts in a class C_c, and let x_c be the number of desired instances (integer) of available parts from this class. Let $\{p_1^r, p_2^r, \cdots, p_n^r\}$ be the set of n relative preferences specified by the n respective genes. The set of n absolute preferences $\{p_1^a, p_2^a, \cdots, p_n^a\}$ is computed through normalization, and is given by

$$p_i^a = \frac{p_i^r}{\sum_{j=1}^{n} p_j^r}$$

Using these absolute preferences an additive partition of the integer x_c is computed as the set $\{x_{c_1}, x_{c_2}, \cdots, x_{c_n}\}$, where $x_{c_i} = \texttt{round}(p_i^a x_c)$ is the number of desired instances of a certain part from its supply source. The function $\texttt{round}(\cdot)$ generates an integer closest to its argument such that $x_c = \sum_{j=1}^{n} x_{c_j}$.

Each row of the matrix after the topmost row corresponds to a manufacturing resource, and a gene at location (r, c) is a weight that signifies

[1] The combinatorial complexity of the problem arises from its structure including the assignment constraints.

the preference of assignment of the entity of column c to the entity of row r.

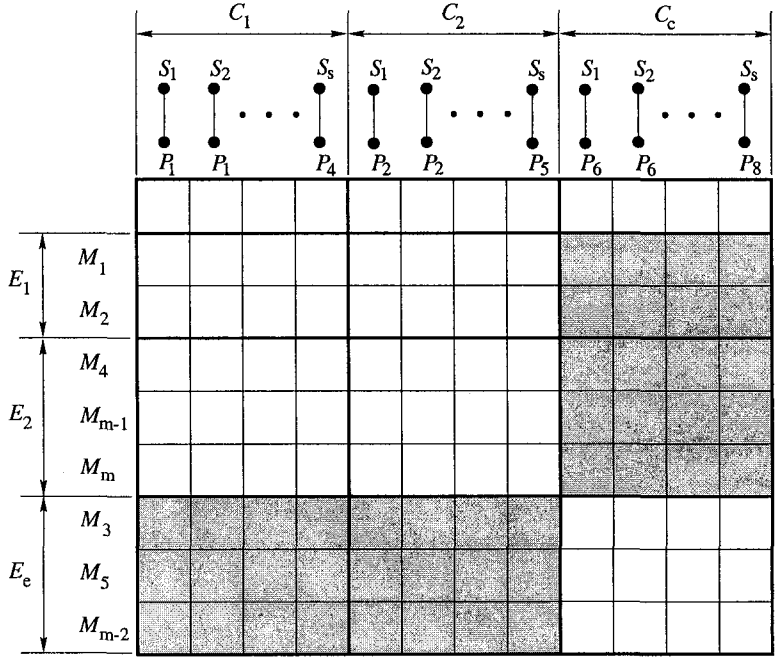


Fig. 6.8 Representation for the general design-supplier-manufacturing decision problem.

Assume the computation of x_{c_i}, which specifies a desired number of instances of a part from a supply source. The next problem is the determination of the distribution of these parts across equivalent manufacturing resources in some set E_e. The solution to this problem is similar to the one described earlier, but now the absolute preferences are computed over a fixed column and variable rows.

For the example shown in Figure 3.1, the part classes C_1, C_2 are not assignable to manufacturing resource class E_e, and the part class C_c is not assignable to manufacturing resource classes E_1, E_2. Therefore, the genes corresponding to these assignments (shown shaded in Figure 6.8) are ignored in any subsequent objective computations.

The representation, which consists of a two dimensional matrix of real numbers in the space $[0, 1]$ serves as a very convenient means to specify assignments when it is used in combination with the partitioning and as-

signment heuristic based on relative preferences. The principal advantage of this representation is that all genomes generated using this data structure correspond to valid partitions and assignments.

6.4.2 *Evolutionary Operators*

Given a population of sample genomes, the evolutionary algorithm selects a parent set using proportional selection. Each parent generates two offspring through Gaussian mutation; the first offspring is generated using a large standard deviation of 0.5 while the second offspring is generated using a tenth of the standard deviation (0.05). The parent and two offspring compete and the fittest survives, and this facilitates a coarse and fine search in the same step. The two dimensional matrix representation could easily support a recombination operation wherein sub-matrices from two or more parents are spliced to create offspring. However, the implementation uses only the gene mutation operation to introduce variation. Also, a fixed population size of 200 is used in the simulations.

6.4.3 *Test Problem Objective*

This is the problem of a "best matched" assignment of parts to machines such that the tool head in a machine has minimum difficulty handling a part. For this it is desired that parts be well matched to machine tool heads in terms of their relative sizes. Let $S(P_j) \in [0,1]$ denote the size of the part P_j, and let $T(M_i) \in [0,1]$ denote the tool head size of machine M_i. The cost of assigning n_{ij} instances of part P_j to machine M_i is computed as $n_{ij}^2(T(M_i) - S(P_j))^2$. Given m machines and p parts, the total cost of assignment to be minimized, assuming each part is assignable to any machine, is computed as the coupled nonlinear objective

$$\sum_{i=1}^{m}\sum_{j=1}^{p} n_{ij}^2(T(M_i) - S(P_j))^2$$

In case some parts are not assignable to some machines, the suggested assignments are ignored in the objective computations.

6.4.4 *Algorithm Performance*

Three test assignment problems (A, B, C) with progressively increasing numbers of search variables are generated. "Assignment Problem" A corre-

sponds to a problem with two parts and four machines (10 search variables), while "Assignment Problem" B corresponds to a problem with four parts and eight machines (36 search variables), and "Assignment Problem" C corresponds to a problem with eight parts and eight machines (72 search variables). For each of these problems the value at the corresponding global optimizer is 0, and this is achieved by setting up each problem such that for a part of a certain size, there exists at least one machine with a tool head of the same size.

In the centralized algorithm implementation all search variables can be simultaneously manipulated. In a distributed implementation the search variables are partitioned among three nodes. For each of the three example problems the distributed implementation uses Coordination Policy–C (see Section 5.2.3.3) with three random samples generated per coordination iteration.

The simulation parameters for the searches are: Network delay $d = 10$, Local access delay $l = 1$, Number of nodes $p = 3$, Coordination frequency (for the distributed algorithm) $f = 0.1$, and Population sizes: $m_c = 200, m_d = 50$.

Figure 6.9 shows the average (and standard deviation) convergence performance over 20 trials of the evolution of the mean fitness of the population distribution of the centralized algorithm, and the evolution of the fitness of the global iterates of the distributed algorithm. Figure 6.10 shows the same comparison with respect to time requirements of the respective algorithms. The distributed search corresponding to this problem consistently achieves the global optimum in a time-efficient manner, and this reinforces the observation that the distributed computation has a better time-performance advantage when network communication delays are considered.

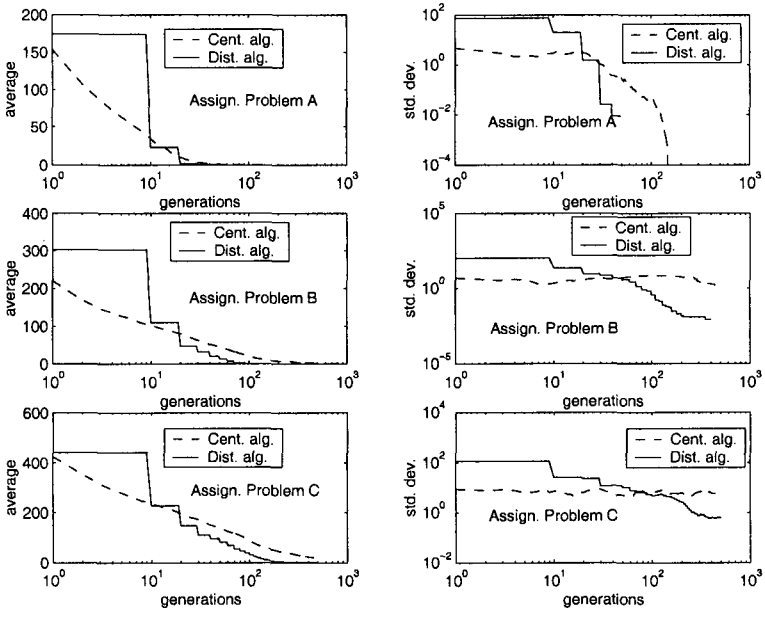


Fig. 6.9 Convergence performance over 20 trials of the centralized and distributed algorithms applied to the example parts–machines assignment problems.

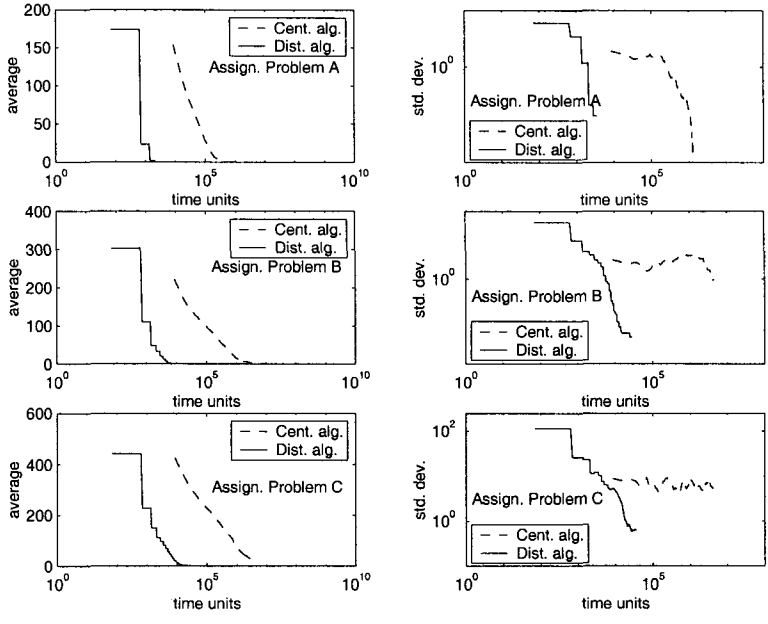


Fig. 6.10 Time performance over 20 trials of the centralized and distributed algorithms applied to the example parts-machines assignment problems.

Chapter 7

Coevolutionary Virtual Design Environment

7.1 Introduction

A principal focus in Chapter 3 is the formulation and analysis of a class of distributed decision problems arising in integrated manufacturing planning, the formulation for which is developed as a set of coupled nonlinear assignment problems. Another principal focus of that chapter is the identification and evaluation of alternative algorithms suitable for global optimization of problems in this class. This evaluation, based on the structure and requirements of the problem class supports the application of evolutionary optimization techniques. Chapters 4, 5, 6 respectively develop theoretical foundations and evaluate two competing approaches to evolutionary optimization–centralized optimization, and distributed coevolutionary optimization. The discussions in these chapters firstly highlight the versatility of evolutionary optimization techniques, and secondly highlight the computational attractiveness of the coevolutionary method for distributed decision problems such as the design-supplier-manufacturing planning decision problem. In these distributed decision problems, multiple logically interrelated decision resources are physically distributed, and information for global decision-making using these resources is available from several network-distributed databases. Therefore, in this environment, network access and database access delays become principal factors that influence time requirements for the optimization process, and in turn affect the suitability of an optimization approach.

The focus of the present chapter is an applications oriented integration of the developments from earlier chapters. This chapter describes the algorithms, architecture and implementation of the coevolutionary decision-making framework, the Coevolutionary Virtual Design Environ-

ment (CVDE). In this framework, an electronic exchange of design, supplier, and manufacturing information facilitates a concurrent, cooperative, network-efficient exploration of complete integrated planning alternatives (*virtual designs*) that are evaluated against an objective function based on cost and time models. These models rely on information available from network-distributed databases. The principal computational entities in the CVDE are distributed evolutionary agents and mobile agents that generate and execute queries among distributed applications and databases to support a global optimization of design, supplier, and manufacturing planning decisions for printed circuit assemblies.

While there are several published articles on the application of agent-based paradigms to manufacturing planning, scheduling, and control problems (e.g. [Sikora and Shaw, 1997; Yu and Huang, 2001]), and evolutionary computation has been applied to solve various manufacturing problems (see a review in [Dimopoulos and Zalzala, 2000]), the combination of evolutionary computation and agent architectures for integrated and optimal network-based planning is a recent development (e.g. [Chao *et al.*, 2002; Hsieh, 2002]).

Section 7.2 describes the printed circuit assembly design-manufacturing application domain, and the application-specific assumptions. Section 7.3 presents evolutionary optimization techniques applicable to design-supplier-manufacturing planning for printed circuit assemblies. This section describes the representation used, objective function and models, evolutionary operations specific to the centralized and distributed coevolutionary optimization techniques, and architectures of the centralized and distributed implementations. Section 7.4 first presents two CVDE implementations–one that physically resides in the memory space of a single processing unit but executes over a simulated network layer, and the other that is physically distributed over several processing units connected over a campus network. While the implementation that executes over a simulated network uses event-driven models for computing communication and access delays, and is very attractive[1] as a testbed for experimental evaluation of algorithm performance, the implementation that executes over a real network serves both as a realistic prototype and as a testbed for evaluating real network-based execution. Finally, the method for generating the underlying data for design-manufacturing planning is described.

This chapter is based in part on material that appears in [Subbu and

[1] From the perspective of execution speed.

Sanderson, 2003b]. Earlier versions have appeared in [Subbu *et al.*, 1998a; Subbu *et al.*, 1998b; Subbu *et al.*, 1998c; Subbu *et al.*, 1999; Subbu and Sanderson, 2001b; Subbu and Sanderson, 2001a].

The following chapter in this book presents a detailed evaluation of the centralized and distributed coevolutionary decision-making frameworks, identifies principal factors that affect their network-based performance, and analyzes the requirements for applicability of the computational models to distributed network-enabled design and manufacturing organizations.

7.2 Application Domain

Design and manufacturing of printed circuit (PC) assemblies is typically realized in three distinct but highly coupled stages. First, a design (string of parts) and suppliers for the parts constituting the design are specified along with a specification of the layout of the parts on a printed circuit board. Next, a fabricator of the printed circuit board is selected. Finally, a manufacturing facility that can assemble the product given a part string and circuit board, is selected. These stages are highly coupled for decisions made at the design stage have maximum impact on product cost, product realization time, and manufacturing, and manufacturing choices have maximum impact on design, product cost, and product realization time. Design and manufacturing decisions affect choices of parts that are assigned to a design, selection of suppliers who will supply the parts, and selection of manufacturing resources that can produce the design.

Traditionally, however, design and manufacturing tasks are executed sequentially with little or no interaction between the phases, and choices are typically made using the experience base in an organization. This results in numerous time consuming iterations and design-supplier-manufacturing solutions that are not anywhere close to optimal in terms of time, cost, or a trade-off function of both time and cost. Moreover, as suppliers and manufacturers increase in number, the combinatorial complexity of optimal selection grows rapidly. An integrated planning method thus assumes importance in this environment as a powerful means to improve competitiveness of products. Such planning methods respect the inherent coupling among the decision stages, and seek to select those plans that result in lower overall cost and lead-time to realization.

Figure 7.1 shows the nature of the integrated design, supplier, manufacturing planning problem for printed circuit assemblies, a formal model

for which is presented in Section 3.3. In this domain, parts, suppliers, and

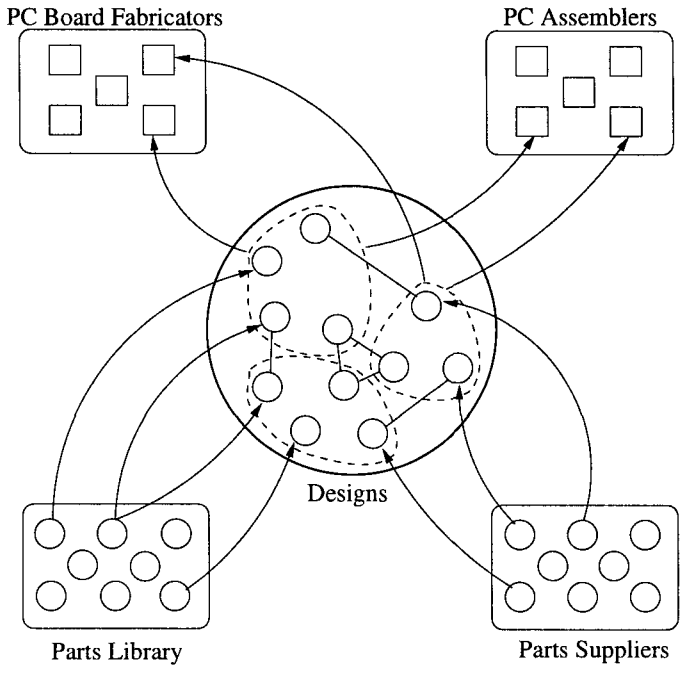


Fig. 7.1 Nature of the integrated design, supplier, manufacturing planning problem for printed circuit assemblies. Lines with arrowheads indicate assignments. Identical parts in various designs have solid lines between them.

manufacturing decision resources are logically interrelated, and information for global decision-making using these resources is available from multiple network-distributed databases.

7.2.1 *Configuration of the Networked Environment*

Figure 7.2 shows a high-level configuration of the networked environment that consists of several logical clusters of network nodes and a product design node. Nodes in a logical cluster correspond to a class of functionally equivalent entities, and in general are physically distributed over a network. In this environment, three logical clusters of network nodes are considered:

- *Parts Distributor Nodes:* Each node in this cluster corresponds to a parts distributor (parts warehouse) that stocks parts from several man-

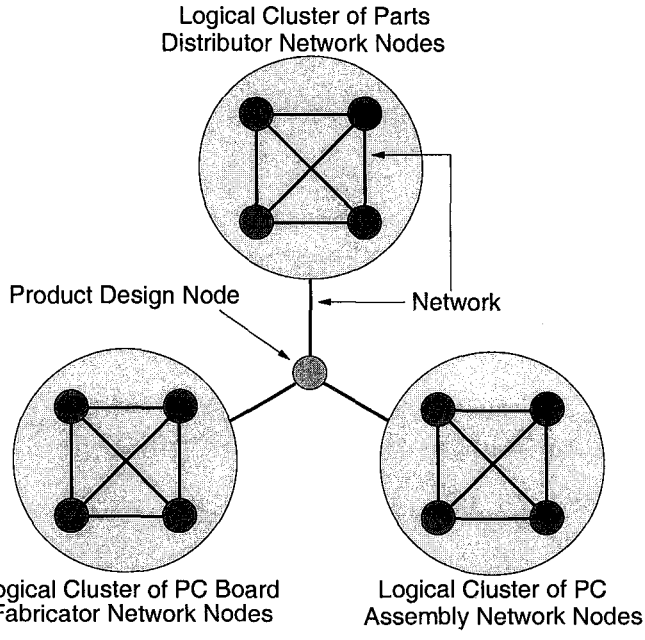


Fig. 7.2 High-level configuration of the networked environment for integrated printed circuit assembly planning.

ufacturers.

- *Printed Circuit Board Fabricator Nodes:* Each node in this cluster corresponds to a printed circuit board manufacturer that may have several alternative board manufacturing lines, each of which is capable of manufacturing printed circuit boards.
- *Printed Circuit Assembly Nodes:* Each node in this cluster corresponds to a manufacturing facility with alternative manufacturing lines, each of which is capable of manufacturing printed circuit assemblies.

Each of the nodes in the above three logical clusters has a locally resident database that stores information specific to the node. A parts distributor node's database stores information on parts and their functionally equivalent alternatives, part characteristics, their costs and lead times for availability. A printed circuit board fabricator node's database stores information that reflects the capabilities of the corresponding fabrication lines, and a printed circuit assembly node's database stores information on the available manufacturing resources within lines, their capabilities, process costs and delays.

In addition, there is a *product design* node that generates functional specifications that serve as partial templates for virtual designs. These templates specify the required parts equivalence classes (module types) and their respective instances for realizing a specific printed circuit board product.

While the search at a parts distributor node is over the space of functionally equivalent designs that correspond to a functional specification, and is achieved by selecting alternative parts and suppliers for those parts, the search at a printed circuit fabricator node is over the space of available board fabrication resources, and the search at a printed circuit assembly node is over the space of available assembly resources.

7.2.2 *Application-Specific Assumptions*

This section lists several assumptions specific to printed circuit assembly planning that influence the organization and implementation of the decision-making environment.

It is assumed that for each module type listed in a functional specification, there are multiple part alternatives that can satisfy the requirement. Part alternatives may differ not just by their supply sources, costs and lead times, but also by their physical characteristics. A commonly occurring physical variation is *packaging*, whereby the same part may be available in multiple package variations. This type of variation has a significant impact on manufacturing processes. Another commonly occurring variation is through *function subsumption*, whereby a given function occurring in a certain part occurs similarly in its part variant along with additional available functions. This type of variation has the potential to affect the physical size of a part, which in turn has consequences in manufacturing. The important issue is that variations in part characteristics and their supply sources have coupled and nonlinear affects on manufacturing, product cost, and product realization time.

An important assumption is made regarding the availability of a printed circuit layout generator. Ideally, this generator would take as input a parts string and a specification of the electrical interconnectivity between the parts, and output a layout on a printed circuit board. However, the discussion in this book assumes the existence of a model-based system that can identify the principal characteristics of the resulting printed circuit board given a design.

Another important assumption is made regarding the manufacturability

of printed circuit board products. It is assumed that any generated printed circuit board is manufacturable at any of the available fabrication lines, and printed circuit assemblies can be assembled at any of the available assembly lines. The rationale is that rather than generating infeasible production options, production options are allowed to incur cost and time penalties consistent with the degree of infeasibility, through the objective function.

A single functional specification is used as a starting point to search over the coupled space of equivalent designs, printed circuit board fabricators, and printed circuit assemblers. However, in principle, one could implement an augmented search that not only searches over equivalent designs corresponding to a single functional specification, but also searches over the space of functional specifications itself. To realize such a system one would require a mechanism (application) that resides at the product design node and generates a space of equivalent functional specifications.

A functional specification is an abstraction at the level of part module types. If it is possible to generate equivalences at a higher level of abstraction (sub-system level), one could implement a much broader search. An example is used to illustrate this point. Assume that a sub-system in a printed circuit board requires one instance of a specific integrated circuit to perform a function. Let there exist a sub-system level alternative wherein the same function can be realized using purely discrete components (resistors, capacitors, transistors, etc.). Then these two sub-system alternatives are considered to have equivalent (not identical) functional specifications from the perspective of the module types they encompass.

7.3 Evolutionary Optimization

This section presents centralized and distributed coevolutionary optimization techniques applicable to design-supplier-manufacturing planning for printed circuit assemblies. First, an application-specific representation particularly suited to this problem is presented. Next, the objective function and embedded models used for evaluating alternative planning decisions are discussed. Finally, the centralized and distributed algorithms and their respective architectures are described.

7.3.1 *Representation*

The template appropriate for representing a virtual design for printed circuit assembly planning is an array of n integers, also called a genome. Each of the first $n-2$ integer slots (genes) encodes a discrete choice from the set of available parts in a part-equivalence-class. The gene at position $n-1$ encodes a discrete choice from the set of board fabrication lines, and the gene at position n encodes a discrete choice from the set of manufacturing lines. The options space of the entire planning problem is adequately represented by the data structure shown in Figure 7.3. In this structure,

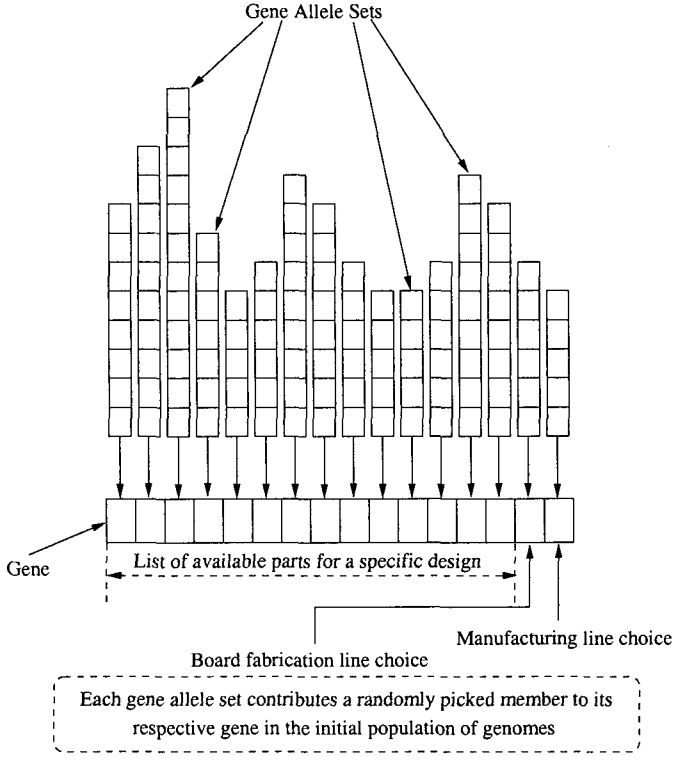


Fig. 7.3 Representation for the printed circuit assembly problem.

each gene allele set is a set of integers representing discrete options along a dimension. A genome (array of genes) is constructed by randomly picking alleles (values) from corresponding gene allele sets. This representation implicitly resolves all constraints defined in the problem formulation, and so

issues of constraint satisfaction do not arise. This allows one to work with an unconstrained problem within the bounds defined by the representation data structure.

7.3.2 *Evaluation and Models*

This section describes the decision resources and models including those that are probed for evaluating alternative virtual designs.

7.3.2.1 *Evaluation*

The overall objective for the printed circuit assembly optimization problem follows the descriptions in Sections 3.2.2, 3.2.3, and 3.3. In the objective function to be minimized, $\psi(C^T, T^T) = C^T e^{(T^T - \alpha)/\beta}$, the values for α and β are set at 10. The constant α acts as a threshold beyond which total time is heavily penalized, and the constant β serves as a time scaling constant. While several other trade-off functions of total cost and total time can be created and used, a threshold-based exponential weighting of the total time serves to emphasize the time factor compared to the cost factor when total time exceeds the threshold. In design and manufacturing applications, bringing a product to market faster is often more important than its initial cost, and this objective function serves to capture that intuitive idea through a mathematical expression.

7.3.2.2 *Models*

Several model-based transformations are utilized in the computation of the overall objective function to be minimized. Figure 7.4 is a high-level view of the embedded models that are discussed below from a conceptual perspective. The internals of the models are described in Appendix B.

- *Part:* Part characteristics are described in Appendix B.1.
- *Design:* A design is an aggregate entity whose characteristics are derived by a transformation of the characteristics of its constituent parts. The *Parts Cost* and *Maximum Parts Lead Time* objective function components are outputs of this entity. Design characteristics are described in Appendix B.2.
- *PCB Gen.:* This represents a printed circuit board generator, which is a model-based transformer that outputs a set of printed circuit board (*PC Board*) characteristics given a set of design characteristics. This is

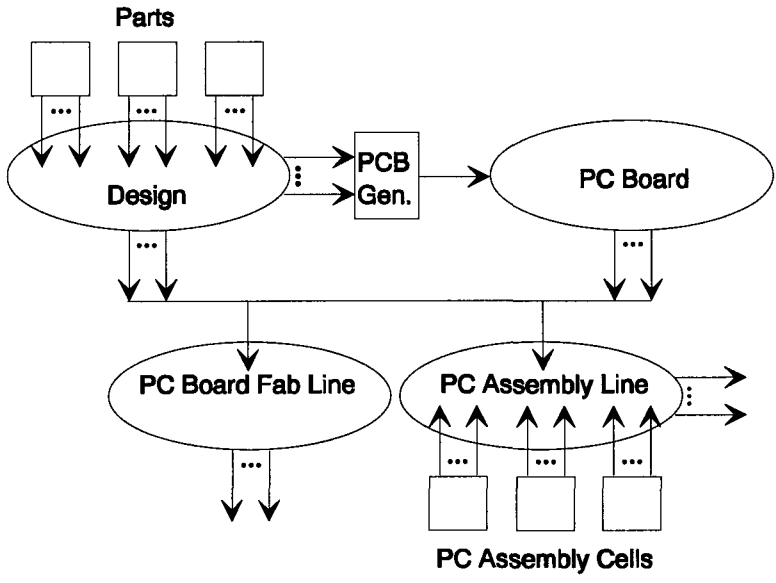


Fig. 7.4 High-level view of models embedded in the objective function. An arrow cluster associated with any entity represents a set of characteristics.

described in Appendix B.3.

- *PC Board Fab Line:* The characteristics of the printed circuit board fabrication line entity are described in Appendix B.4. A board fabrication line takes as input the characteristics of a design and an associated printed circuit board. The *Overhead Cost* and *Overhead Lead Time* objective function components are outputs of the printed circuit board fabrication line entity. These outputs are respectively the fabrication cost and fabrication time of the printed circuit board. The rationale is that assembly cannot commence until a printed circuit board is fabricated, and so board fabrication time is treated as an overhead lead time.
- *PC Assembly Cell:* The characteristics of a printed circuit assembly cell are described in Appendix B.5. An assembly cell realizes a specific function in the overall assembly process.
- *PC Assembly Line:* This entity takes as input the characteristics of a design and an associated printed circuit board, and these are processed by each of the constituent cells in a manufacturing line. The *Manufacturing Cost* and *Manufacturing Time* objective function components

are outputs of the printed circuit assembly line entity.

7.3.3 *Centralized Optimization*

In a centralized optimization mode, an evolutionary algorithm is resident at one of the network nodes (see Figure 7.2), and the other network nodes participate in the search by providing information on request. In this mode, a search over the full space of planning decisions takes place at one node, while the other nodes merely respond to network-based information requests. The product design node launches the evolutionary search by creating an *evolutionary agent* at one of the network nodes, and provides it a design functional specification and location information of the other network nodes.

7.3.3.1 *Architecture*

Figure 7.5 shows the architecture of the centralized system wherein the evolutionary search is executed by the *evolutionary agent* resident at one of the network nodes. The evolutionary agent implements an evolutionary search by initializing with appropriate information that allows for decision-making, and generating and executing queries among distributed applications for evaluating virtual designs created during the search process. Each network node that supports the evolutionary search has a locally resident node-specific application that is accessed by the evolutionary agent during the virtual design evaluation process. Each such application is either a server for the local database at that node or is a model that executes based on the information gleaned from the local database. For instance, the application at a parts distributor node serves as a conduit for accessing information on parts, while the application at a printed circuit board assembly node is a relevant model that takes as input a design and associated printed circuit board, and outputs manufacturing cost and manufacturing time corresponding to a selected manufacturing line.

7.3.3.2 *Algorithm*

The evolutionary agent initiates the evolutionary search by searching each of the network nodes for available decision resources and encoding references to the discrete choices in the appropriate gene allele sets of its representation data structure. A certain gene allele set corresponding to a certain part module type stores an encoded list of equivalent parts available from

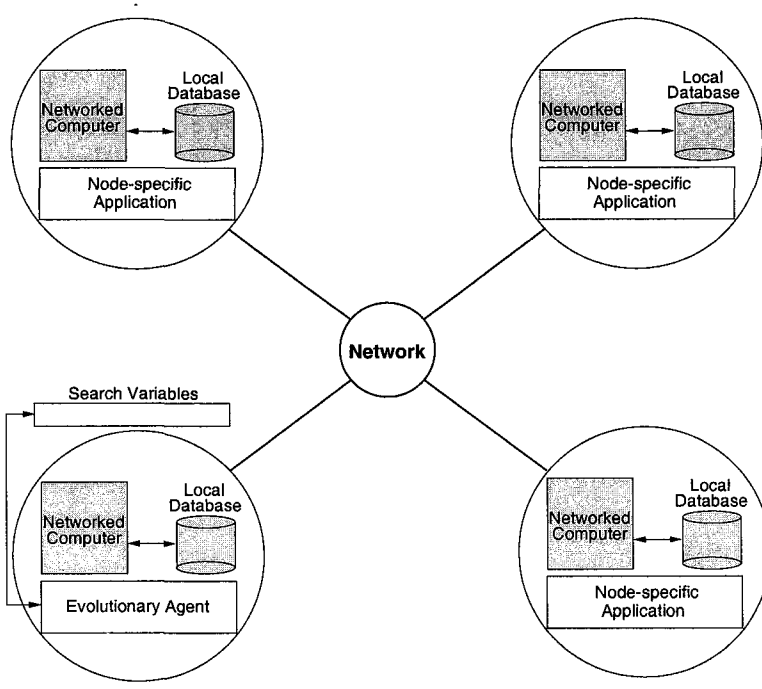


Fig. 7.5 Architecture of the centralized system.

all parts distributor nodes, and in a similar fashion the gene allele sets corresponding to all other module types store information on their respective sets of available equivalent parts. The gene allele set corresponding to printed circuit board fabrication maintains an encoded list of printed circuit board fabrication lines at all printed circuit board fabrication nodes, and the gene allele set corresponding to printed circuit assembly maintains an encoded list of printed circuit assembly lines at all printed circuit assembly nodes.

Once the template data structure is complete, the evolutionary algorithm embedded in the evolutionary agent creates and evaluates virtual designs, using proportional selection and random mutation operations to evolve virtual designs. Each newly created virtual design is evaluated based on a series of network-based information requests.

The discussion in Chapter 4 guides the implementation of the evolutionary algorithm embedded in the evolutionary agent. The convergence theory in that chapter is developed in the space of reals to facilitate a

tractable and compact mathematical analysis, and serves as a quantitative foundation for describing evolutionary algorithm behavior. From the perspective of the representation for the printed circuit assembly planning problem, discrete selections from each of the gene allele sets are made in order to complete a sample genome, making it an inherently discrete problem. However, a discrete optimization problem may be posed as a suitably encoded real space optimization problem if necessary, and so the theoretical developments in real space optimization are relevant in the context of discrete optimization. For example, discrete problems are often solved by solving their corresponding real space relaxations. Also, consider the discussion in Section 6.4 where the search variables in the representation (for the general design-supplier-manufacturing planning problem) are bounded real intervals $[0, 1]$, but are interpreted in a discrete fashion to solve an inherently discrete problem.

7.3.4 *Distributed Coevolutionary Optimization*

In a distributed coevolutionary optimization mode, an evolutionary search takes place at each of the network nodes, and the nodes must cooperate to jointly search over the full space of planning decisions. In this mode, a node participates in network-based communication only during coordination operations. At other times the evolutionary algorithm at a node works with locally available information. The product design node launches the evolutionary search at all the network nodes by creating an *evolutionary agent* at each node.

7.3.4.1 *Architecture*

Figure 7.6 shows the computational components of a network node, each of which is composed of a networked computer, a local database, and an evolutionary agent and several mobile agents that execute on the networked computer.

An *evolutionary agent* at a network node performs the following functions:

- Implements a local evolutionary algorithm that searches over the subspace corresponding to locally available information.
- Initializes with appropriate information that allows for local decision-making.
- Generates and executes queries on the local database.

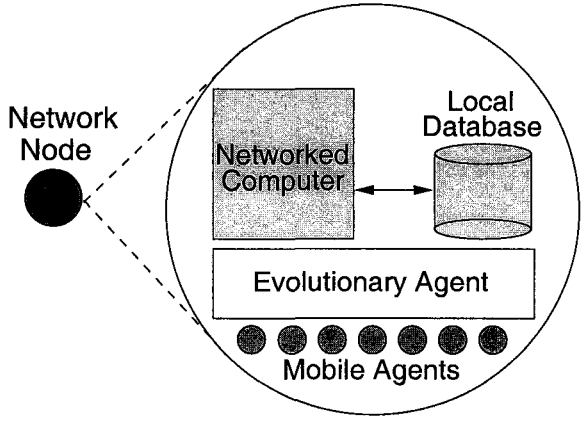


Fig. 7.6 Components of a network node in distributed coevolutionary computation.

- Coexists in a pool of evolutionary agents, and participates in coordinating the global computation through interactions with other evolutionary agents and mobile agents.

An evolutionary agent at a node i is able to solve only the subproblem $\min_{z \in \mathcal{X}_i} \psi(z|\overline{x}_i)$, and an intercommunication with the other distributed evolutionary agents updates its secondary variable set $\overline{x}_i$. However, for the printed circuit assembly planning problem, there is a nonlinear coupling between all the variables, and for a local variable assignment $z \in \mathcal{X}_i$ an evaluation $\psi(z|\overline{x}_i)$ cannot be completed without network-based information requests from other nodes. This interdependency between subproblems arises because it is not possible to neatly decompose the overall evaluation into a set of subproblem evaluations such that there is an exact fit between a subproblem evaluation and location of information. This problem is important in distributed computation and also appears in the context of multiagent systems research [Lesser, 1999]. As a novel solution to this problem, and to facilitate a network-efficient computation of the objective function at each node, *mobile agents* are introduced in the architecture. These mobile agents migrate to nodes during intercommunication operations carrying updated information from the nodes they migrate and provide the missing computational functionality at a node. For instance, an agent that migrates from a printed circuit assembly node to a parts distributor node carries with it a model of resources and information of a specific assembly line (at the printed circuit assembly node), such that when the evolutionary agent at

the parts distributor node selects an alternative set of parts, the associated cost and time of assembly at that manufacturing line can be computed.

The mobile-agent architecture allows a computationally attractive alteration to the computational delay model for the distributed computation. In Section 5.3.2, the time delay per generation of evolution of the distributed algorithm, t_d, is given by $t_d = lm_d + [4d + (2d + l)c](p - 1)f$, where the term $4d$ models two network access cycles required to collect and transmit results, and the term $(2d + l)c$ models the cost of accessing a network node for evaluating c joint computations. When the cardinality c of the randomly created set $\{x_g'\}$ increases (resulting in a higher number of network-based evaluations during every coordination operation), the advantage to distributing the computation decreases. However, an architecture based on mobile agents facilitates the elimination of the additional penalty due to network-based evaluations under the following condition:

- The elements of the set $\{x_g'\}$ which represent randomly created virtual designs are created (by the node performing the coordination) purely by recombination of virtual designs already obtained from the other nodes. From an architectural perspective, obtaining a virtual design from a node results in obtaining corresponding mobile agents. Then, one would already have locally available mobile agents that can assist in locally evaluating new virtual designs created by recombination, and as a consequence no network-based evaluations are required ($c = 0$).

When the node performing the coordination operation updates the other nodes, copies of the corresponding mobile agents simultaneously migrate to these nodes. Therefore, this update operation is efficiently performed using a multi-cast, and the $4d$ term above reduces to approximately $2d + \alpha$, where α is the per node overhead for the multi-cast operation. If d is large then $t_d \approx lm_d + 2d(p - 1)f$.

7.3.4.2 *Algorithm*

The evolutionary algorithm embedded in each evolutionary agent is similar in function to the evolutionary algorithm of the centralized search. However, an evolutionary algorithm at a node utilizes a representation that encodes only the locally available decision resources in its representation data structure.

The computational model assumes that the p variable blocks are non-overlapping and that there is a tight nonlinear coupling amongst them with

respect to the nonlinear objective function $\psi(\cdot)$. Under this assumption, a communication operation involves all p nodes. However, if the problem were such that there is no or only a loose coupling between certain search variable chunks, then a communication need not involve all p nodes, and only those nodes that are coupled to one another need to communicate frequently. The network-based communication overhead may be significantly reduced by this strategy. This would be especially relevant if the objective function is decomposable into constituent parts, each part of which is dependent only on a certain variable chunk.

The computational model also assumes that communication between nodes is synchronous, and that if local information at the nodes is varying that rate of variation is much slower than the rate of search, effectively allowing an evolutionary search to work under the assumption of a static global fitness landscape.

The evolutionary algorithm embedded in each evolutionary agent is similar in function to the evolutionary algorithm of the centralized search. However, an evolutionary algorithm at a node utilizes a representation that encodes only the locally available decision resources in its representation data structure.

In the context of a distributed global search, the coordination operation is critical, since a coordination operation essentially provides an updated view of the local information from a certain node to another node where that information is not available locally. An important feature of a coordination operation is its ability to allow the coevolutionary search to explore the same global space of planning options as the centralized algorithm in spite of the fact that each evolutionary agent is only able to search a subspace of the global search space. The following chapter presents several coordination schemes that help realize this. Presented below is the "splicing" operation, a component of the coordination operation that becomes necessary when there is more than one node per logical cluster of network nodes (see Figure 7.2).

Recall that nodes in a logical cluster correspond to a class of functionally equivalent resources. As a consequence the subproblems solved at nodes in a cluster are different in spite of being functionally similar. The subproblems are different because of differences in local resources at each node, so each node in a logical cluster searches over a smaller space of planning decisions. The coevolutionary algorithm has no direct means to search the full space of planning decisions. A centralized algorithm on the other hand compiles a list of all available decision resources at all nodes and is able to

explicitly search the full space of planning decisions. In order to provide a coevolutionary algorithm the means to aggressively explore the full space of planning decisions, an information splicing operation is introduced whose principal function is to stochastically combine information from nodes. This stochastic information splicing may be viewed as a "crossover" operation that combines information across nodes.

When there are multiple nodes in each logical cluster of network nodes, the evolutionary algorithm components (in the distributed implementation) in some of the nodes are unable to achieve convergence to the overall global solution. This is an inevitable problem in a distributed algorithm implementation since there may be some nodes within a logical cluster that simply do not have the local resources that are advantageous from a global perspective. However, this problem is tractable since those nodes within a logical cluster that have globally advantageous decision resources will achieve good global convergence similar to the centralized algorithm, which has access to all decision resources.

7.4 Simulation Environments

7.4.1 *CVDE Implementations*

This sections describes two implementations of the coevolutionary virtual design environment. In these implementations, the underlying network is assumed to be a "fully connected" graph $G = (N, E)$, where N is the set of network nodes and E is the set of network edges. In other words, it is assumed that it is possible to directly access a network node from any other network node. Both implementations are highly modular, based on the object-oriented programming paradigm [Booch, 1994; Stroustrup, 1991], and can be executed in either a centralized mode or a distributed coevolutionary mode.

The first implementation is developed using the C++ [Stroustrup, 1991] programming language, and executes completely in the memory space of a single processing unit, but over a simulated network environment (see Figure 7.7). This implementation uses event-driven computation of network access and local access delays.[2] Since this implementation executes completely on a single processing unit while simulating a distributed system, the overall computations are rapid making this an attractive environment for

[2]Uniform access delays are assumed over the network environment.

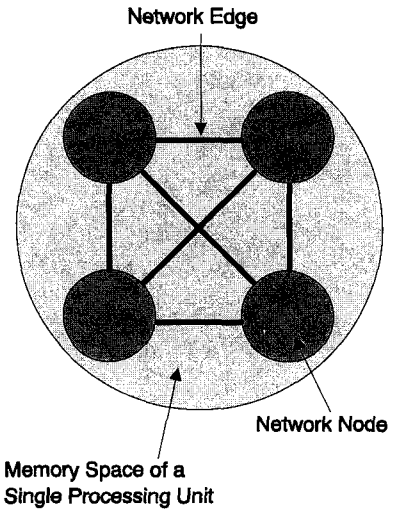


Fig. 7.7 Simulated network environment CVDE implementation that executes in the memory space of a single processing unit.

evaluation of algorithm performance. Due to the implementation's execution speed, it is feasible to develop experiments based on multiple trials on several stochastically generated printed circuit assembly planning problems. Such simulation is a powerful approach that facilitates rapid prototyping of various ideas and solutions, and allows better exploration of the space of algorithms. An important advantage of this simulation environment is that it provides a controlled environment for evaluating and analyzing alternatives, and facilitates an understanding of the scope of these alternatives including coordination schemes. The algorithms found promising in a simulation phase are implemented to execute over a real network.

The second implementation is developed using the Java (SUN Microsystems Inc.) programming language and executes over multiple processing units distributed over a campus computing network (see Figure 7.8). This implementation is based on the use of the Voyager [voy, 2000] object request broker as the underlying distributed communications environment. Voyager serves as a middle-ware layer that provides a location-transparent and standardized environment for execution of the Java modules. A significant advantage is that Voyager simplifies the task of remote enabling applications modules by automatically adding this feature at run-time, and supports the inter-node migration of modules. The latter feature is an important requirement for realizing the "mobile agents" feature in the dis-

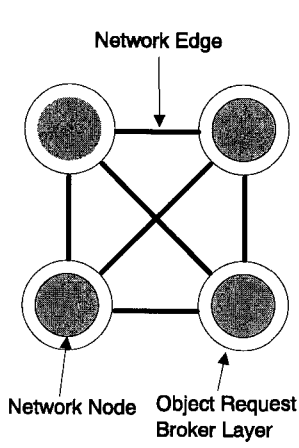


Fig. 7.8 Real network environment CVDE implementation that executes over multiple distributed processing units.

tributed architecture. The principal advantages of this implementation are that it serves as a realistic prototype and as a testbed for evaluating real network-based execution of the system.

In this environment, there may be some nodes and network edges that are faster or slower than the others, so some nodes may complete their local computations faster than others and vice versa. Asynchronous behavior among nodes is avoided by synchronizing the steps of the overall computation.

7.4.2 *Data Generation*

The underlying data for design and manufacturing planning scenarios are created using `datagen`, a simulation environment data generator. This application uses a combination of design and manufacturing data and randomization to generate design functional specifications, and to populate the databases resident at the various network nodes. Design functional specifications are patterned on sample designs provided by Pitney Bowes Inc., parts information are patterned on information available online from Arrow Electronics Inc., and manufacturing data are in part patterned on sample manufacturing resources at Avex Electronics Inc., and Allen-Bradley Corp., and complemented by process information from [Taylor and Graves, 1990].

The inputs to `datagen` are the number of parts distributor nodes, number of printed circuit board fabrication nodes, number of printed circuit assembly nodes, and the number of module types in a design functional

specification. The outputs are parts information in parts databases, board fabrication line specifications in board fabrication node databases, and assembly cell and line specifications in the assembly node databases. From the perspective of size of the search spaces that can be generated, `datagen` is able to generate arbitrarily large search spaces.

Chapter 8

Evaluation and Analysis

8.1 Introduction

A principal focus of this chapter is the performance evaluation of the centralized and distributed coevolutionary decision-making frameworks and developing an understanding of the factors that influence their network-based performance. Another equally important focus is an analysis of the requirements for applying these decision-making frameworks to distributed design and manufacturing organizations.

Section 8.2 observes the nature and evolution of design-supplier--manufacturing planning decisions for printed circuit assembly planning. This observation helps to develop an applications oriented interpretation of the results of the evolutionary planning. Section 8.3 develops a strategy for evaluating the performance of the centralized and distributed coevolutionary decision-making frameworks. Section 8.4 evaluates the decision-making frameworks and identifies the principal factors that influence performance. Next, the CVDE implementation that executes over a real network is utilized to develop an understanding of the delay factors, and a combination of experimental evaluation and models is used to predict the range of expected execution times of the decision-making frameworks for large design problems when the underlying network is the internet. Section 8.5 presents an analysis of the requirements for applicability of the decision-making frameworks to distributed design and manufacturing organizations.

This chapter is based in part on material that appears in [Subbu and Sanderson, 2003b], and earlier in [Subbu and Sanderson, 2001b; Subbu and Sanderson, 2001a].

8.2 Nature and Evolution of Planning Decisions

This section develops an applications oriented interpretation of the nature and evolution of design-supplier-manufacturing planning decisions (virtual designs) for printed circuit assembly planning. First, the performance of sample virtual designs from example design-manufacturing applications is discussed. Next, several examples of the convergence behavior of the centralized and distributed coevolutionary frameworks are presented.

`datagen` described in the previous chapter is used to generate an example printed circuit design-manufacturing application with 10 module types, with a search based over 5 parts distributor nodes, 5 printed circuit board fabrication nodes, and 5 printed circuit assembly nodes. The size of the search space for this problem is in excess of 2.42×10^{22} options. Table 8.1 shows the performance of good and poor virtual designs for this design-manufacturing application. Good virtual designs favor lower cost parts with shorter lead times, and lower manufacturing costs and times, while poor virtual designs have higher cost parts with longer lead times, and higher manufacturing costs and times.

Table 8.1 Virtual design performance (Exp-1).

Variable	Good	Good	Poor	Poor
Parts Cost	80.56	126.23	244.30	202.24
Max. Parts Lead Time	11.37	13.44	17.64	18.47
Board Fab. Cost	2.25	3.67	3.51	5.67
Board Fab. Time	1.13	1.98	1.82	3.05
Manufacturing Cost	3.80	19.74	195.24	413.91
Manufacturing Time	2.60	3.73	68.48	117.38
C^T	86.61	149.64	443.04	621.81
T^T	13.96	17.17	86.12	135.85
$\psi(C^T, T^T)$	128.69	306.50	8.96×10^5	1.82×10^8

Performance of good and poor virtual designs for a design application with 10 module types, 5 parts distributors, 5 printed circuit board fabrication facilities (64 fabrication lines), 5 printed circuit assembly facilities (66 assembly lines).

Another example printed circuit design-manufacturing application with 10 module types is generated, with a search based over 10 parts distributor nodes, 10 printed circuit board fabrication nodes, and 10 printed circuit assembly nodes. The size of the search space for this problem is in excess of 2.65×10^{25} options. Table 8.2 compares the performance of good virtual designs from Table 8.1 to the performance of good virtual designs for the current application. All good designs favor lower cost parts with shorter

lead times, and lower manufacturing costs and times.

Table 8.2 Virtual design performance (Exp-1 and Exp-2).

Variable	Exp-1	Exp-1	Exp-2	Exp-2
Parts Cost	80.56	126.23	94.56	112.65
Max. Parts Lead Time	11.37	13.44	9.47	9.33
Board Fab. Cost	2.25	3.67	2.27	2.53
Board Fab. Time	1.13	1.98	1.14	1.28
Manufacturing Cost	3.80	19.74	5.29	39.03
Manufacturing Time	2.60	3.73	3.00	10.15
C^T	86.61	149.64	102.13	154.21
T^T	13.96	17.17	12.47	19.49
$\psi(C^T, T^T)$	128.69	306.50	130.80	398.17

Comparison of the performance of good virtual designs from Table 8.1 (Exp-1) to the performance of good virtual designs (Exp-2) for a design application with 10 module types, 10 parts distributors, 10 printed circuit board fabrication facilities (107 fabrication lines), 10 printed circuit assembly facilities (119 assembly lines).

Figures 8.1, 8.2, and 8.3 respectively show the time-performance of the centralized and distributed algorithms in 3, 15, and 30 network node environments. The search space sizes respectively are in excess of 2.75×10^{10}, 5.67×10^{16}, and 1.15×10^{19} options. These performance plots show the average over 20 trials of the time-performance (of the best member in each generation) of the centralized computation and the average over 20 trials of the time-performance (of the best member in each generation) of each of the distributed algorithm components.[1] The network environment for these simulations has an $\frac{l}{d}$ ratio of 0.001, and a coordination operation occurs every generation of the distributed search. These figures show that a distributed coevolutionary framework has significant time-performance advantage compared to a centralized framework when network access and local access delays are considered.

When there are multiple nodes in each logical cluster of network nodes, the evolutionary algorithm components (in the distributed implementation) in some of the nodes are unable to achieve convergence to the overall global solution. This is an inevitable problem in a distributed algorithm implementation since there may be some nodes within a logical cluster that simply do not have the local resources that are advantageous from a global perspective. Those nodes within a logical cluster that have globally advan-

[1] An average time-performance plot is achieved by plotting the average performance of the best member in each generation of evolution with respect to the average time delay for the network-based computation until that point.

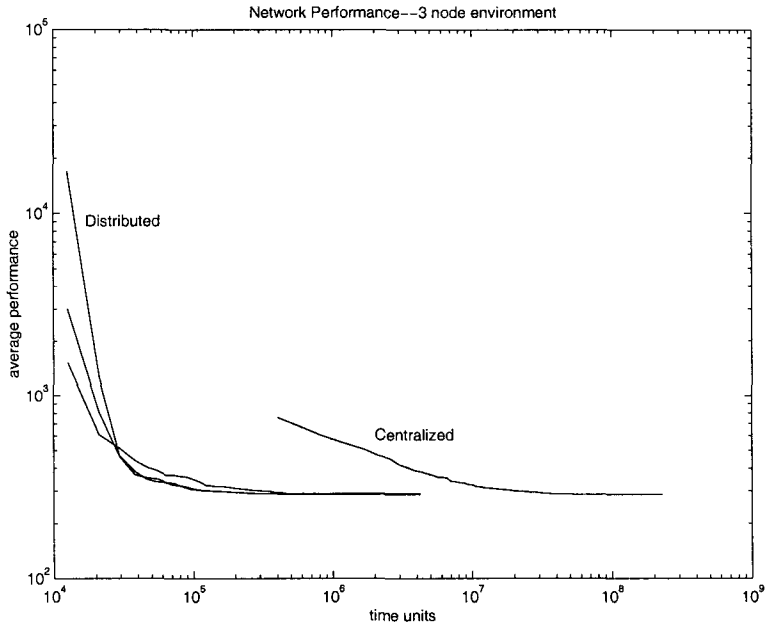


Fig. 8.1 Average time-performance of the centralized and distributed algorithms in a 3 node environment (1 parts distributor node, 1 printed circuit board fabrication node, 1 printed circuit assembly node). $\frac{l}{d} = 0.001$.

tageous decision resources will achieve good global convergence similar to the centralized algorithm, which has access to all decision resources.

8.3 Strategy for Performance Evaluation

This section develops a consistent approach to evaluating the centralized and distributed coevolutionary decision-making frameworks. First, performance evaluation metrics are proposed, and next, algorithm factors of interest are discussed.

It is common practice in algorithm design to evaluate algorithms against a suite of test problems with known optima. Such an evaluation has already been performed in Chapter 6, and while a similar evaluation would be attractive in the current context of design-supplier-manufacturing planning for printed circuit assemblies, these search spaces are very large and nonlinear, so approximations to the global optima are satisfactory. Therefore, the average performance (of the best member in each generation) over 20

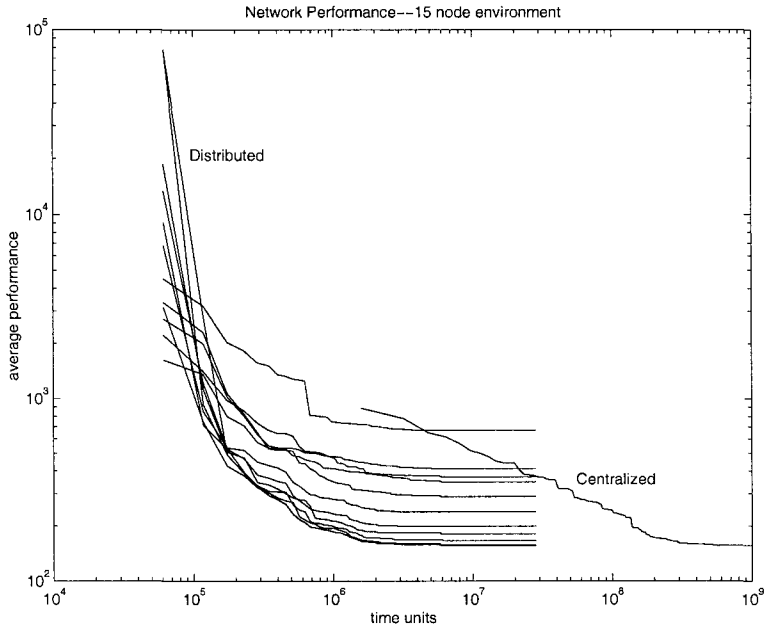


Fig. 8.2 Average time-performance of the centralized and distributed algorithms in a 15 node environment (5 parts distributor nodes, 5 printed circuit board fabrication nodes, 5 printed circuit assembly nodes). $\frac{l}{d} = 0.001$.

trials of a baseline centralized evolutionary algorithm is used as the yardstick against which other algorithms are evaluated. The baseline centralized evolutionary algorithm has the following characteristics:

- Population size is fixed at 100.
- A parent population is generated using proportional selection.
- Each parent generates two offspring through random mutation: the first offspring is generated using a large mutation probability of 0.2, and the second offspring is generated using a tenth of the mutation probability (0.02). The parent and two offspring compete and the fittest survives. The offspring generated in this fashion replace the individuals from the earlier generation.

In order to ensure a fair comparison between algorithms, the average performance (of the best member in each generation) over 20 trials is compared to the performance of the baseline centralized algorithm. Also, each comparison is performed over a space of 10 design-manufacturing planning

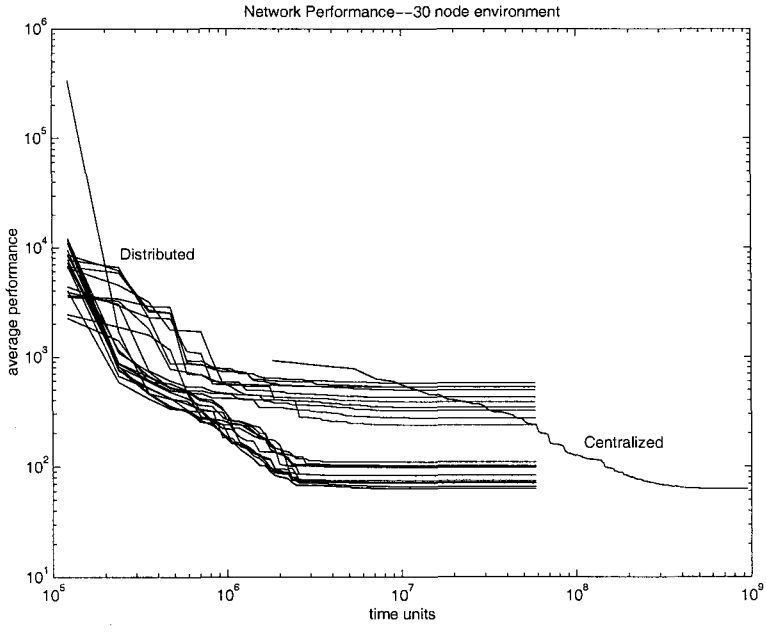


Fig. 8.3 Average time-performance of the centralized and distributed algorithms in a 30 node environment (10 parts distributor nodes, 10 printed circuit board fabrication nodes, 10 printed circuit assembly nodes). $\frac{l}{d} = 0.001$.

scenarios (generated using `datagen`) with number of network nodes in the range $[3, 30]$ (search space sizes are in the range $[2.75 \times 10^{10}, 1.15 \times 10^{19}]$). Therefore, in effect, an algorithm comparison involves 200 trials.

8.3.1 *Performance Metrics*

Network-based performance of algorithms is evaluated based on two metrics discussed below. In this discussion, "performance" refers to average performance (of the best member in each generation) over 20 trials, and "steady-state error" refers to the error between the averaged convergence profile of a given algorithm normalized against the performance of the baseline centralized algorithm at a large-enough generation (generation 500).

- *Percentage Convergence Error (% Err.):* This measures the minimum percentage steady-state error of the distributed evolutionary algorithm components.

- *Percentage Computational Advantage (% Adv.):* This is the ratio of the average time by which the performance of the baseline centralized algorithm comes within 10% of its steady-state performance to the minimum average time by which the performance of any of the distributed algorithm components comes within 10% of the steady-state performance of the baseline algorithm.

8.3.2 *Factors of Interest*

In general it is of interest to investigate the following issues:

- The algorithm characteristics that govern convergence quality.
- The time-performance of an algorithm with a certain characteristic.
- How delays in the underlying network environment affect performance of the algorithms.
- How the algorithm with a certain characteristic sustains the potential benefits of the feature as the problem size increases.

The discussion in Chapter 5 suggests that for distributed coevolutionary algorithms, the coordination operation is critical to global convergence. Therefore, it is of interest to investigate the types of coordination schemes that lead to convergence with low errors. The discussion in Section 7.3.4 highlights the importance of information splicing for distributed algorithms when there are multiple nodes in each of the logical clusters. Information splicing is also the randomization component of the coordination operation, so these evaluations are treated together. Next, it is of interest to know how these schemes sustain any advantage when there is a moderate amount of time-variability in the states of the decision resources. Finally, it is of interest to know the effect of coordination frequency on convergence, and the effect of the network ratio $\frac{l}{d}$ on algorithm performance.

8.4 Performance Evaluation

This section evaluates network-based performance of the centralized and distributed coevolutionary decision-making frameworks, and identifies the principal factors that influence performance. For this purpose, the CVDE implementation that executes over a simulated network is used. Next, an understanding of the nature of real execution delays is developed using the CVDE implementation that executes over a campus network.

8.4.1 *Evaluation Over a Simulated Network*

8.4.1.1 *Coordination and Information Splicing*

Six distributed coordination schemes are evaluated in conjunction with three types of information splicing–(*i*) *No Splicing*, (*ii*) *Uniform Splicing*, and (*iii*) *Single-Point Splicing*. In the implementation of uniform splicing, p vectors of the same dimension are used to create a vector such that each of its coordinates is a random selection from the set of p coordinates along the same dimension. In the implementation of single-point splicing, a pair of parent vectors is selected at random from the set of p vectors, a splice dimension is selected at random, and a new vector is created whose coordinates left of and including the splice dimension come from one parent, and whose coordinates right of the splice dimension come from the other parent. The following helps describe the coordination schemes:

- The network environment has p network nodes.
- $x_g^{(i)}$ is the best vector from node i at generation g.
- $\{x_g^{(i)}\}$ is a set of vectors from node i at generation g.
- $\{x_g{}'\}$ is a set of randomly created vectors at generation g.
- y_g is the vector obtained by combining the best local result portions from each node.

Then, the six coordination schemes are as follows:

(1) *Local:* Create the set $\{x_g^{(1)}, \cdots, x_g^{(p)}, \{x_g{}'\}\}$, and select the best from the set as the new global iterate. When there is no information splicing the set $\{x_g{}'\}$ is empty. Otherwise, the set $\{x_g{}'\}$ consists of p elements created (according to the splicing scheme) from the set $\{x_g^{(1)}, \cdots, x_g^{(p)}\}$.

(2) *Joint:* Create the set $\{x_g^{(1)}, \cdots, x_g^{(p)}, \{x_g{}'\}\} \cup y_g$, and select the best from the set as the new global iterate. The elements of the set $\{x_g{}'\}$ are generated exactly as above depending on the splicing operation used.

(3) *Pool:* Create the set $\{\{x_g^{(1)}\}, \cdots, \{x_g^{(p)}\}, \{x_g{}'\}\}$, and select the best from the set as the new global iterate. Each set $\{x_g^{(i)}\}$ represents $t = 5$ top performers from each node i, and the set $\{x_g{}'\}$ is created as described above from a set of size $t \times p$ rather than a set of size p.

(4) *Elite Local:* Create the set $x_g \cup \{x_g^{(1)}, \cdots, x_g^{(p)}, \{x_g{}'\}\}$, and select the best from the set as the new global iterate. x_g is the previous global iterate.

(5) *Elite Joint:* Create the set $x_g \cup \{x_g^{(1)}, \cdots, x_g^{(p)}, \{x_g'\}\} \cup y_g$, and select the best from the set as the new global iterate.
(6) *Elite Pool:* Create the set $x_g \cup \{\{x_g^{(1)}\}, \cdots, \{x_g^{(p)}\}, \{x_g'\}\}$, and select the best from the set as the new global iterate.

Figure 8.4 shows the average (and standard deviation)[2] of the percentage convergence error with respect to coordination type (rows top $\rightarrow$ bottom : $1 \rightarrow 6$), information splicing scheme (columns), and the abscissa in each plot is the number of generations between nodal intercommunication (communication interval in generations). When no information splicing is

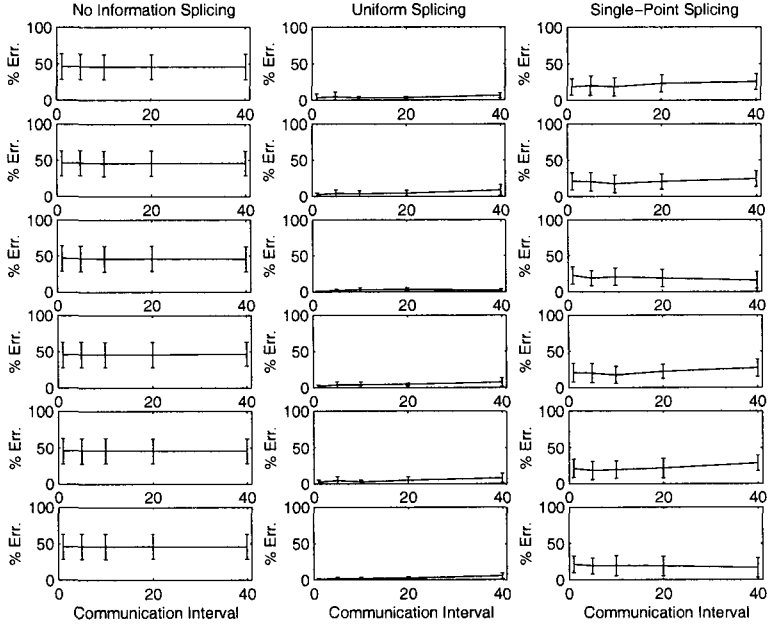


Fig. 8.4 Average (and standard deviation) of the percentage convergence error with respect to coordination type (rows top $\rightarrow$ bottom : $1 \rightarrow 6$), information splicing scheme (columns), and communication interval in generations (abscissa).

used, the percentage convergence error is very high since the distributed algorithm is unable to search over the complete space of planning decisions. Single-point splicing has a lower overall error compared to when no splicing is used, while uniform splicing has the best error performance.

[2]Averaged over the space of test problems.

These figures show that the type of information splicing (incorporation of randomization) has a *first-order effect* on distributed convergence.

Figure 8.5 shows in more detail percentage convergence error plots from the second column in Figure 8.4. Uniform splicing is used in all coordination schemes. The type of coordination scheme has a *second-order effect* on distributed convergence, with *Pool* and *Elite Pool* being the better performers. The coordination scheme *Pool* with uniform splicing gives the lowest percentage convergence error over all communication intervals.

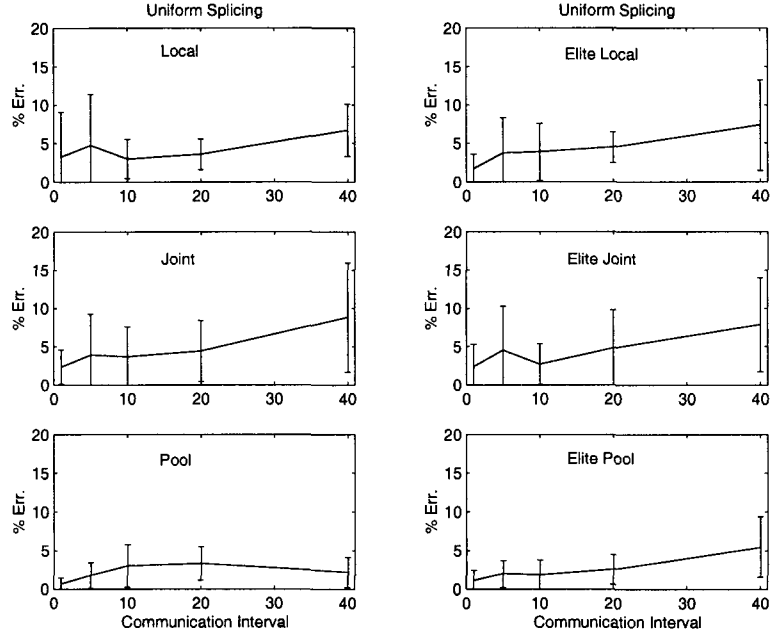


Fig. 8.5 Average (and standard deviation) of the percentage convergence error with respect to coordination type and communication interval in generations (abscissa).

Figure 8.6 shows the percentage computational advantage (for a network environment with ratio $\frac{l}{d} = 0.1$) with respect to coordination type, and the abscissa in each plot is the number of generations between nodal intercommunications. Uniform splicing is used for all coordination schemes. For a network with ratio $\frac{l}{d} = 0.1$ it is more advantageous to communicate every generation than it is to communicate less frequently.

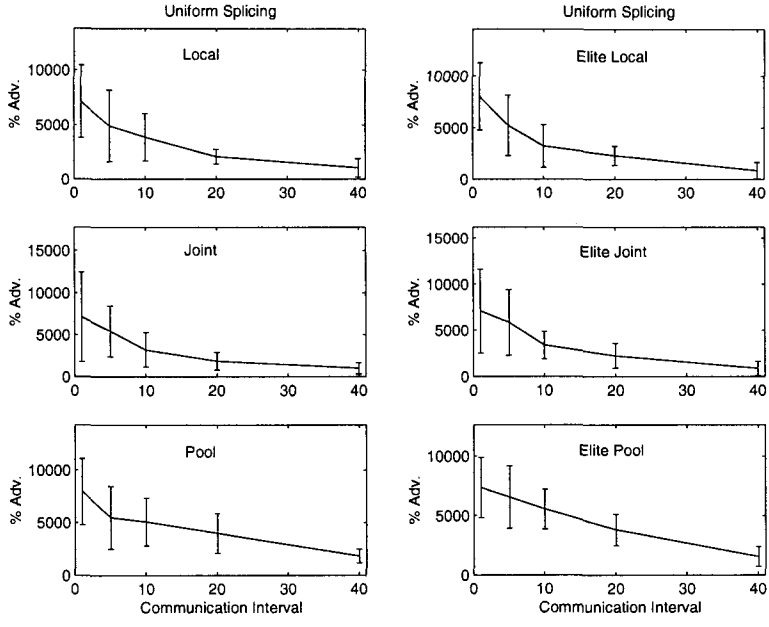


Fig. 8.6 Average (and standard deviation) of the percentage computational advantage with respect to coordination type and communication interval in generations (abscissa). $\frac{l}{d} = 0.1$.

Next, the performance of the coordination and information splicing schemes is evaluated when there is time-variability in the environment: when characteristics of the decision resources change randomly up to 20%(±10%) as a search proceeds. Part costs and lead times are variable, and so are the overheads and efficiencies of printed circuit board fabrication lines and printed circuit assembly lines. A centralized algorithm is able to access the current state of any decision resource, but in the case of the distributed algorithm, current states are available only during coordination operations. In between coordination operations the distributed algorithm works with stale states of the decision resources.

Figure 8.7 shows the average (and standard deviation)[3] of the percentage convergence error (in a dynamic environment) with respect to coordination type (rows top → bottom : 1 → 6), information splicing scheme (columns), and the abscissa in each plot is the number of generations between nodal intercommunication (communication interval in generations).

[3]Averaged over the space of test problems.

The results obtained show consistency with those obtained in a static en-

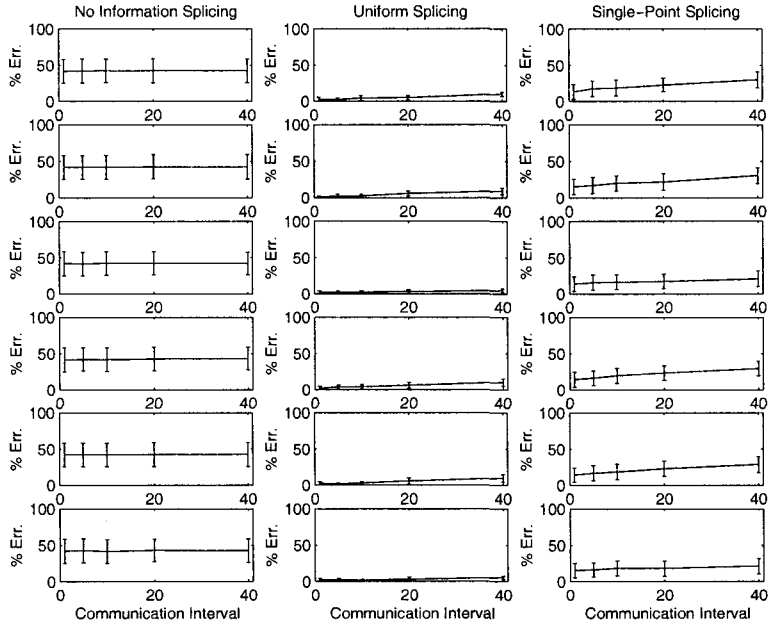


Fig. 8.7 Average (and standard deviation) of the percentage convergence error (in a dynamic environment) with respect to coordination type (rows top $\rightarrow$ bottom : $1 \rightarrow 6$), information splicing scheme (columns), and communication interval in generations (abscissa).

vironment, and uniform information splicing still has a first-order effect on distributed convergence in a moderately dynamic environment.

Figure 8.8 shows in more detail percentage convergence error plots from the second column in Figure 8.7. Uniform splicing is used in all coordination schemes. Though the type of coordination scheme has a second-order effect on distributed convergence, the coordination scheme *Pool* in conjunction with uniform splicing gives the lowest percentage convergence error over all communication intervals.

Figure 8.9 shows the percentage computational advantage (in a dynamic environment, and with a network ratio $\frac{l}{d} = 0.1$) with respect to coordination type, and the abscissa in each plot is the number of generations between nodal intercommunications. Uniform splicing is used for all coordination schemes. These results show consistency with those obtained in a static

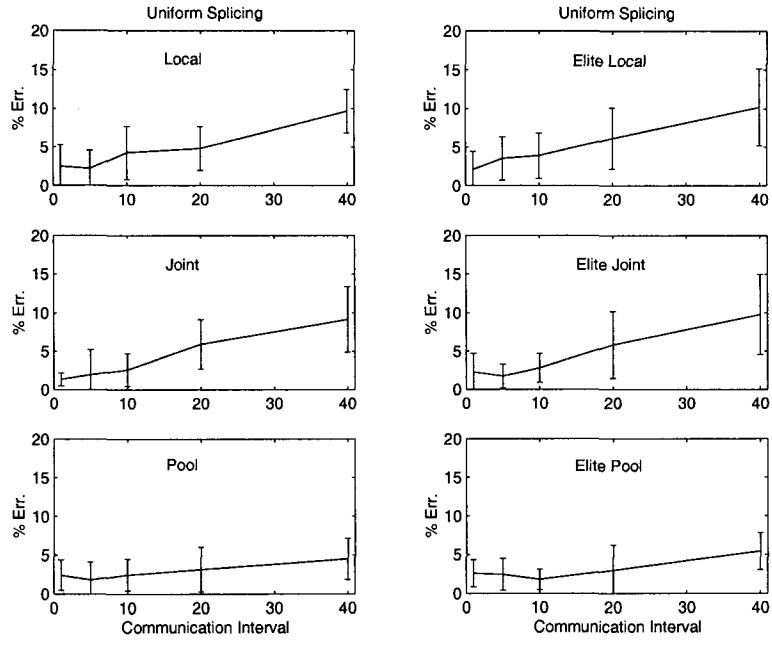


Fig. 8.8 Average (and standard deviation) of the percentage convergence error (in a dynamic environment) with respect to coordination type and communication interval in generations (abscissa).

environment.

8.4.1.2 *Access Delays and Coordination Frequency*

Now, the computational advantage of the distributed coevolutionary framework is evaluated with respect to the communication interval and network ratio $\frac{l}{d}$. The distributed algorithm in these evaluations uses the *Pool* coordination scheme with *Uniform Splicing*, determined above as the combination that results in low percentage convergence errors over all communication intervals. These evaluations do not introduce dynamic variations in the states of the decision resources.

Figure 8.10 shows the average (and standard deviation) of the percentage computation advantage as a function of the network ratio $\frac{l}{d}$ and the communication interval. A higher intercommunication frequency (lower on the communication interval scale) is advantageous in environments with higher network ratios (> 0.02), corresponding to the cases when local ac-

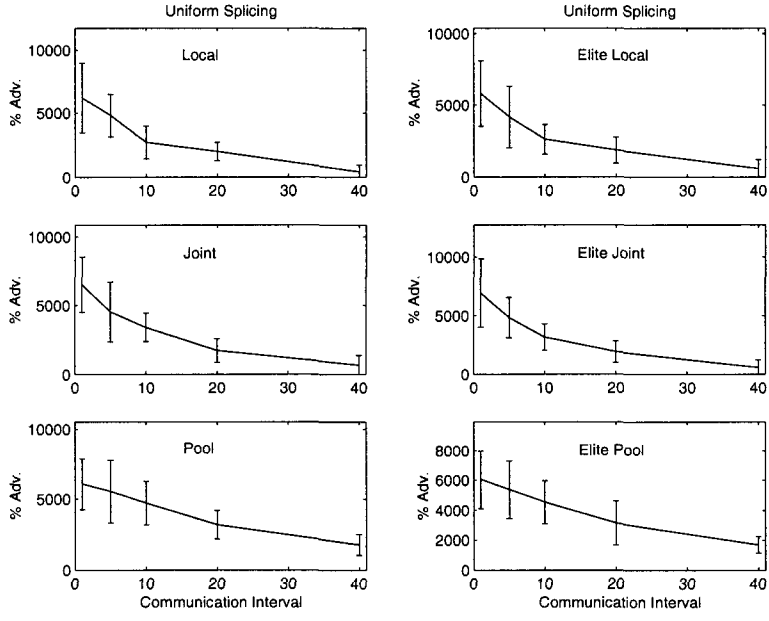


Fig. 8.9 Average (and standard deviation) of the percentage computational advantage (in a dynamic environment) with respect to coordination type and communication interval in generations (abscissa). $\frac{l}{d} = 0.1$.

cess delays are a larger fraction of the network access delays. An important observation is that for network environments with small delay ratios (< 0.02) it is systematically more advantageous to communicate less frequently. Also, as the network ratio gets very large, the advantage of distributing the computation reduces.

8.4.2 *Evaluation Over a Real Network*

This section evaluates the centralized and distributed coevolutionary decision-making frameworks when they execute over a local-area cluster of SUN-Ultra-10 machines situated within a few meters of one another (in the same room) and interconnected over a wired 100Mbps network. First, several examples of the convergence behavior of the centralized and distributed coevolutionary frameworks in this environment are presented. Next, an understanding of the factors causing the delays is developed, and this information is used to compute expected delays when the decision-making frameworks are implemented over the general internet. For this, a

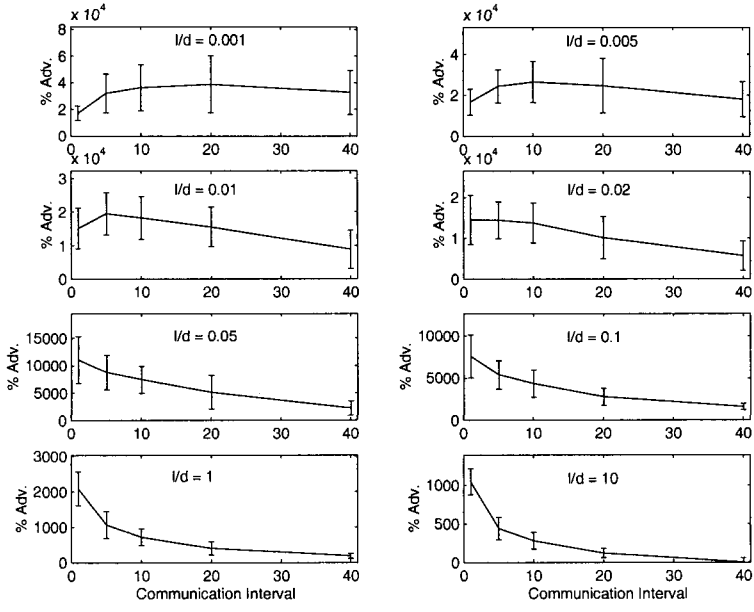


Fig. 8.10 Average (and standard deviation) of the percentage computational advantage with respect to network ratio $\frac{l}{d}$ and communication interval in generations (abscissa).

combination of experimental evaluation and models is used.

An example printed circuit design-manufacturing application is generated with 10 module types with the search respectively based over 3, 6, and 9 node environments, wherein each node corresponds to a unique machine. Figures 8.11, 8.12, and 8.13 respectively show the time-performance of the centralized and distributed algorithms in the 3, 6, and 9 node environments. The coordination operation in a distributed search occurs every 10 generations of evolution, and the search space sizes respectively are in excess of 4.95×10^{13}, 7.54×10^{17}, and 5.23×10^{19} options. These figures show the significant computational advantage of a distributed coevolutionary framework even in an environment with a very fast underlying network connection.

This preliminary evaluation indicates that there is a significant penalty to intercommunication in spite of a fast physical network, and this is because most of the communications delay in a fast local area network is due to the processing time required to send and receive messages.

To develop a quantitative understanding of the factors contributing to

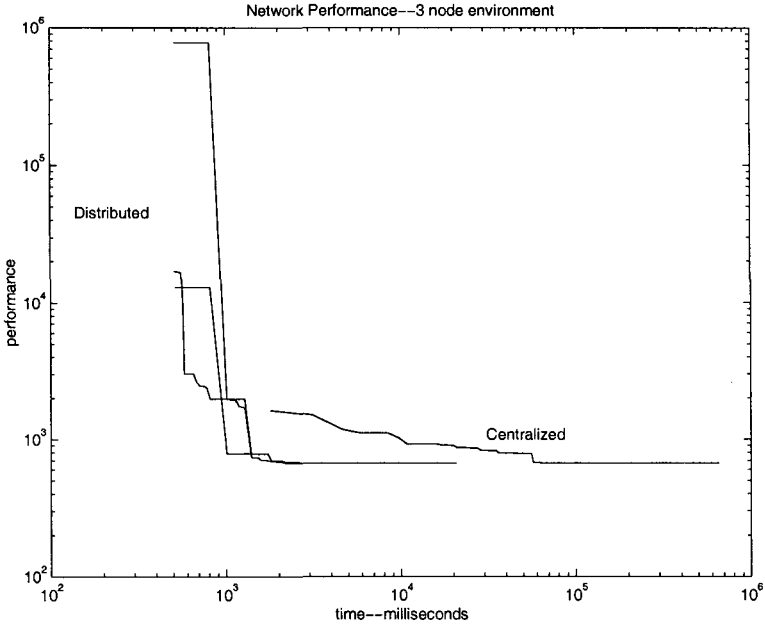


Fig. 8.11 Time-performance of the centralized and distributed algorithms in a 3 node environment (1 parts distributor node, 1 printed circuit board fabrication node, 1 printed circuit assembly node).

delays, a simplified delay network delay model (8.1) due to [Gray, 1988] is used as a foundation. In this model of network communication delay, the time required to send and receive a message is computed as

$$d = t_{tr} + t_{cpu} + \frac{m_s}{b_w} \tag{8.1}$$

where t_{tr} is the speed of light delay to transmit one bit, t_{cpu} is the message processing time of the sender and receiver nodes, m_s is the size of the message in bits, and b_w is the bandwidth of the communication medium in bits per second. This simplified model does not assume more complex factors such as queuing delays and transmission delays caused by passage through intermediate switching equipment, and in a widely distributed environment such factors are significant. Nonetheless, this model is applicable in widely distributed environments for order of magnitude delay computations [Gray, 1988], and for the current network environment (small-diameter fast 100Mbps) it is acceptable to completely ignore these more complex factors.

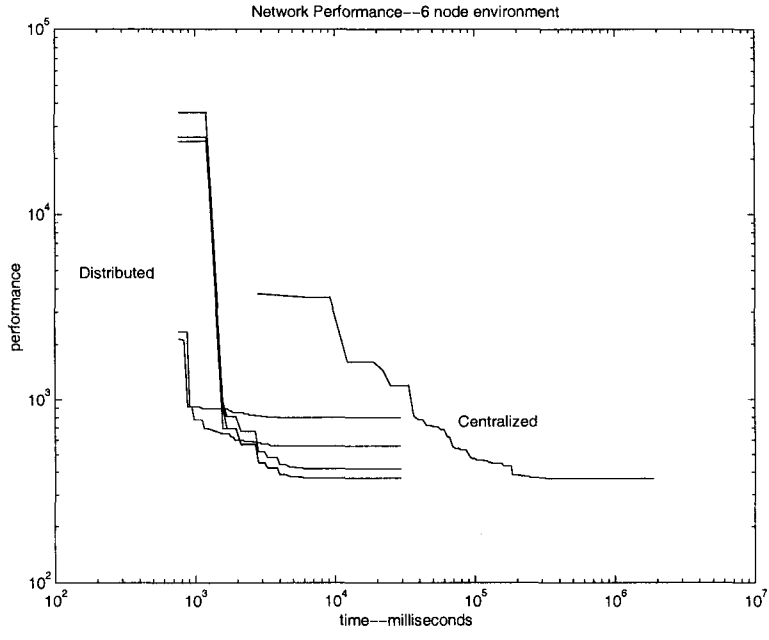


Fig. 8.12 Time-performance of the centralized and distributed algorithms in a 6 node environment (2 parts distributor nodes, 2 printed circuit board fabrication nodes, 2 printed circuit assembly nodes).

In this small-diameter fast network environment, the factor t_{tr} is a very small fraction of a microsecond, and can be ignored. Therefore,

$$d = t_{cpu} + \frac{m_s}{b_w} \tag{8.2}$$

Three experiments (see Figure 8.14) are now setup to measure the message processing overhead, the size of the messages, and the corresponding network delay d. In all three experiments a centralized evolutionary algorithm with a population size of 100 executes at Node-1, while Node-2 and Node-3 respond to information requests. Node-1 is a parts distributor node, Node-2 is a printed circuit board fabrication node, and Node-3 is a printed circuit assembly node. In this setup, the time per generation of evolution is given by the model (8.3), which is a version of the model from Section 5.3.1 adapted to reflect an uneven distribution of local variables.

$$t_c = nlm_c + (2d + l)(p - 1)m_c \tag{8.3}$$

In the expression above, n is the number of local variables per member of

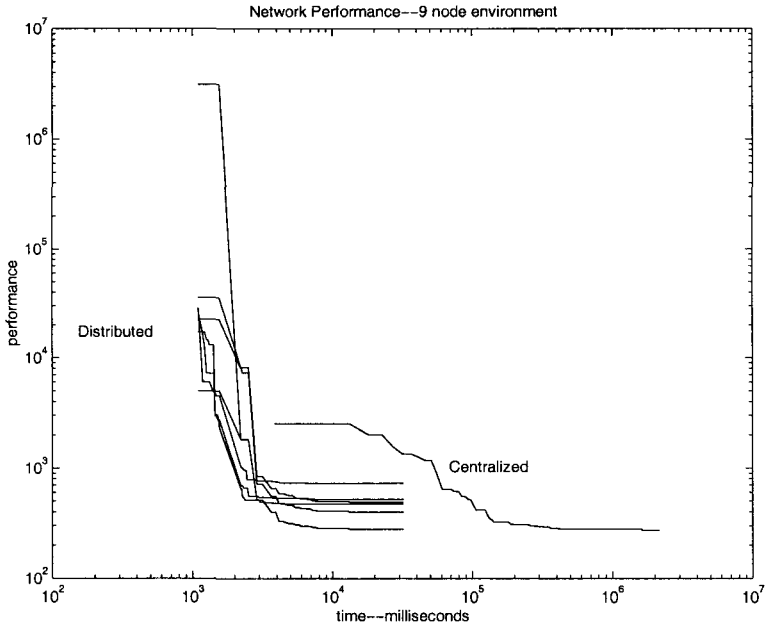


Fig. 8.13 Time-performance of the centralized and distributed algorithms in a 9 node environment (3 parts distributor nodes, 3 printed circuit board fabrication nodes, 3 printed circuit assembly nodes).

the population at Node-1, l is the local access delay, p is the number of nodes, and m_c is the population size. Since Node-1 is a parts distributor node, there are multiple local accesses at that node per member of the population.

In *Experiment-1*, the three nodes correspond to the same physical machine and share a single port (network address) on the machine. Therefore, the network delay $d = 0$, and (8.3) is reduced to

$$\begin{aligned} t_c^1 &= nlm_c + l(p-1)m_c \\ &= 100l(n+2) \end{aligned} \tag{8.4}$$

In *Experiment-2*, the three nodes correspond to the same physical machine but have dissimilar port assignments. In this setup there is no on-wire transmission of information, but since network addresses are dissimilar, messages must be processed by sender and receiver nodes respectively as though they are to be sent over a wire and received from over a wire.

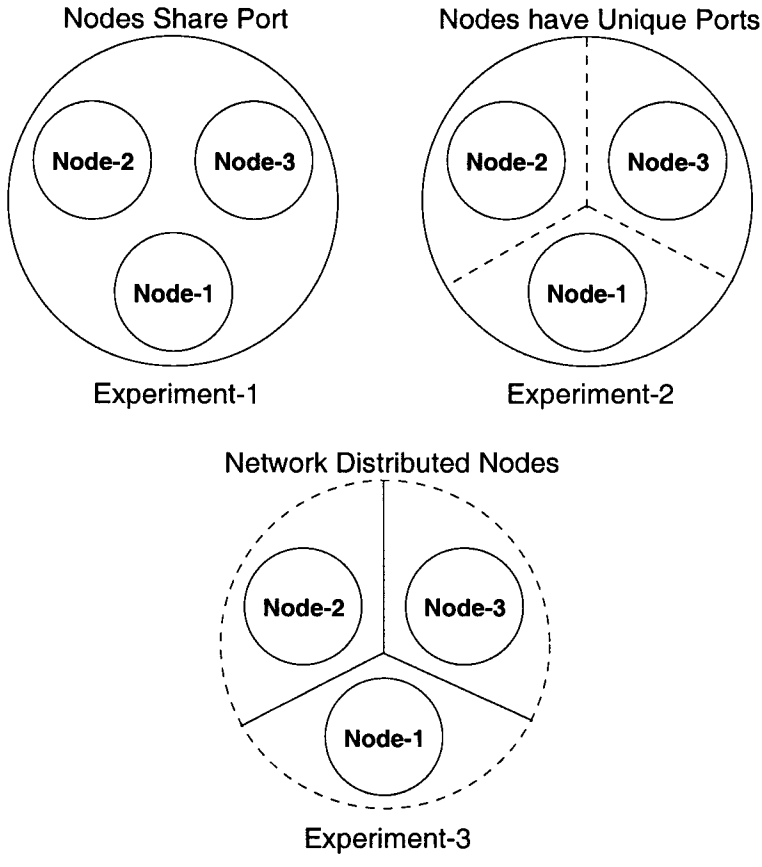


Fig. 8.14 Experimental setup to measure the message processing overhead, the size of the messages, and the corresponding network delay d.

Therefore, $d = t_{cpu}$, and (8.3) is reduced to

$$\begin{aligned} t_c^2 &= nlm_c + (2t_{cpu} + l)(p-1)m_c \\ &= 100l(n+2) + 400t_{cpu} \\ &= t_c^1 + 400t_{cpu} \end{aligned} \tag{8.5}$$

In *Experiment-3*, the three nodes correspond to different machines, and in this setup there is message processing and transmission over a network.

Therefore, $d = t_{cpu} + \frac{m_s}{b_w}$, and (8.3) is reduced to

$$\begin{aligned} t_c^3 &= nlm_c + \left[2\left(t_{cpu} + \frac{m_s}{b_w}\right) + l\right](p-1)m_c \\ &= 100l(n+2) + 400t_{cpu} + 400\frac{m_s}{b_w} \\ &= t_c^2 + 400\frac{m_s}{b_w} \end{aligned} \tag{8.6}$$

Figure 8.15 shows the experimental measurements corresponding to *Experiments*-1, 2, and 3. t_c^1, t_c^2, and t_c^3 have nearly linear growth rates with respect to n. Using these experimental measurements and (8.4), (8.5), and (8.6), average values (over the range of n) for the factors l, t_{cpu}, and

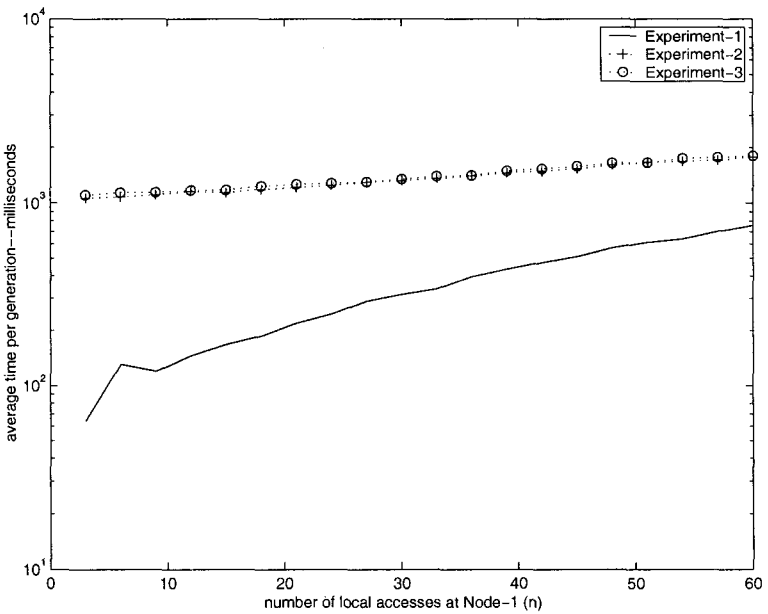


Fig. 8.15 Average time per generation as a function of local accesses at Node-1, for *Experiments*-1, 2, and 3.

m_s are computed (see Table 8.3).

These results confirm that most of the communications penalty in a fast local area network is due to the processing time required to send and receive messages. Also, message sizes are dependent on two principal factors–the volume of information exchanged and the per access overhead of the middle-

Table 8.3 Network delay factors.

Variable	Average Value
l	0.1094ms
t_{cpu}	2.5239ms
$\frac{m_s}{b_w}$	0.0898ms
m_s	$\approx$ 9196bits

Average values for the delay factors assuming a network bandwidth $b_w = 100$Mbps.

ware layer. Therefore, in any distributed communications it is important to reduce both the volume of information exchanged and the number of accesses.

The principal benefit of the above results is that it provides a means to develop reasonable predictions for the time delays of the centralized and distributed algorithms when the underlying network environment is the internet and larger-scale problems are solved.

8.4.2.1 *Expected Time Performance in Realistic Settings*

This section uses the delay factor computations above to predict the range of expected delay times of the centralized and distributed algorithms in realistic settings.

A problem size range of $[300, 3000]$ decisions with an even distribution of decisions among 30 network nodes is assumed. Therefore, each node performs $[10, 100]$ decisions based on locally available information, and the range of l (based on Table 8.3) is approximately $[1, 10]$ ms. Message sizes are assumed to lie in the range $[10^4, 10^5]$ bits. In local-area network settings, the factor t_{cpu} does not grow very significantly with message size as it typically takes a certain number of fixed-duration machine instructions to send and receive a message. However, in wide-area network settings this factor could easily triple [Gray, 1988]. Therefore, it is assumed that t_{cpu} falls in the range $[5, 10]$ ms. A conservative estimate of the current bandwidths over the general wired internet is the range $[1, 10]$ Mbps. Also, a distribution radius of $[500, 3000]$ miles is assumed, which maps to an ideal t_{tr} (8.1) of approximately $[3, 16]$ ms. However, over public carriers, t_{tr} is approximately a millisecond per hundred miles [Gray, 1988], and the range $[5, 30]$ ms is used instead. The estimate for network delay d is computed using (8.1) and multiplying the result by a factor of 4 to account for potential high rates of network utilization, which is frequently the case in shared wide-area networks. The predictions for d in Table 8.4 are similar to the recent measurements by [Sikdar *et al.*, 2000] for TCP transfers of files with similar

Table 8.4 Network delay factor ranges.

Variable	Best Case	Worst Case
l	1ms	10ms
t_{cpu}	5ms	10ms
m_s	10^4bits	10^5bits
b_w	10Mbps	1Mbps
t_{tr}	5ms	30ms
d	44ms	550ms

Ranges of the delay factors in realistic settings.

sizes transmitted over similar distances.

In the future, as internet bandwidths and available computing power improve significantly, the bottleneck factor in network delays will be the factor t_{tr} rather than t_{cpu} and $\frac{m_s}{b_w}$ for messages of reasonable size–the lower bound of the factor t_{tr} being determined by the laws of physics for the distances involved.

Table 8.5 uses the best case and worst case values of l, d, and t_{cpu} from Table 8.4 to compute the best case and worst case execution times in seconds per generation of evolution of the centralized and distributed algorithms in a 30 node environment. The time per generation of evolution of the centralized algorithm, t_c, is computed by $t_c = lm_c + (2d+l)(p-1)m_c$, and the time per generation of evolution of the distributed algorithm, t_d, is given by $t_d = lm_d + (2d+\alpha)(p-1)f$ (see Section 7.3.4), where f is the normalized frequency of coordination, and α is the per node overhead in a multi-cast operation. Based on the discussion above, it is satisfactory to assume $\alpha = \frac{t_{cpu}}{2}$, and so $t_d = lm_d + (2d + \frac{t_{cpu}}{2})(p-1)f$.

Table 8.5 Best and worst case execution times.

Variable	Best Case	Worst Case
t_c	258.2s	3220s
t_d $(f = 1)$	2.7s	33s
t_d $(f = 0.1)$	0.4s	4.2s

Best case and worst case execution times per generation of evolution for the centralized and distributed algorithms (with population sizes $m_c = m_d = 100$) in a 30 node environment with an even distribution of decision variables.

Based on the results in Table 8.5, a centralized algorithm would take nearly 36 hours in the best case to complete 500 generations of evolution, while the distributed algorithm would take nearly 23 minutes in the best case to complete the same number of generations if coordination were to take place every generation. In the worst case, a centralized algorithm would take well over 18 days to complete 500 generations, while the dis-

tributed algorithm that communicates every generation would perform the computation in nearly 4.6 hours, and if communication were to take place every 10 generations, then the computation would take nearly 35 minutes to complete.

As a test, an example printed circuit design-manufacturing application is generated with 100 module types with the search respectively based over a 22 node (20 parts distributor nodes, 1 printed circuit board fabrication node, 1 printed circuit assembly node) distributed environment (small diameter 100Mbps campus network). It is assumed that the search at any parts distributor node occurs over the full set of module types based on locally available resources, so the search at a node is over a subspace of the global space of nearly 10^{250} options. While the centralized algorithm takes

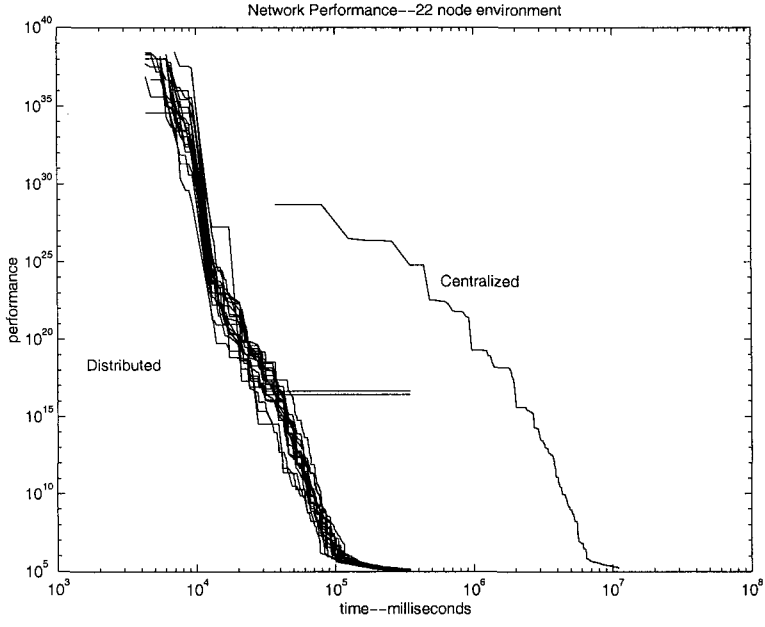


Fig. 8.16 Time-performance of the centralized and distributed algorithms in a 22 node environment (20 parts distributor nodes, 1 printed circuit board fabrication node, 1 printed circuit assembly node).

nearly 44 seconds per generation, the distributed algorithm communicating every 10 generations takes less than 0.7 seconds per generation, and these results are consistent with the nature of the predictions.

8.5 Applicability Analysis of the Frameworks

This section presents an analysis of the centralized and distributed decision-making frameworks from the perspective of their applicability to network-enabled design and manufacturing organizations. Principally, the characteristics of these environments and the computational requirements for implementing these frameworks are discussed.

8.5.1 *Characteristics of the Computational Environment*

The computational environment in which the decision-making frameworks must function is characterized by several large databases distributed over the general internet. Such databases contain information on parts, suppliers, and manufacturing resources and processes, which are logically interrelated from the perspective of integrated design, supplier, and manufacturing planning. The discussion in Section 8.4.2.1 helps develop an appreciation for the network delays (d) to be expected in these environments. The local access delay factor (l) in this discussion is a smaller fraction of the network delay factor since the application modules in the network nodes are able to access information from local in-memory databases. Commercial databases however are generally very large, and processing times for these systems can easily exceed several seconds per query [Chen, 1996; Hohenstein *et al.*, 1997], unless sophisticated caching and parallel processing mechanisms are incorporated. As a result, it is important to carefully consider the requirements for implementing the decision-making frameworks in a time-efficient manner.

Another important issue that arises in the design of such decision-making frameworks is proprietary information security. The success of these frameworks depends on the unhindered availability of information on decision resources, and processes. Often, the perceived potential danger of exposing proprietary information, even when such information sharing is mutually beneficial, poses serious hurdles to the implementation of cooperative information sharing ventures.

8.5.2 *Implementation Strategies*

This section considers proposals for implementing the decision-making frameworks, and the computational requirements for these proposals. In this discussion, the fundamental assumption is that increasing the process-

ing power and local response time characteristics at a network node is vastly inexpensive compared to realizing an equivalent improvement in wide-area network performance.

8.5.2.1 *Proposal-A*

Assumptions

Each network node that supports the evolutionary search has a locally resident application that is accessed by the evolutionary agent during the virtual design evaluation process. An application is either a server for the local database at that node or is a node-specific model that executes at a higher level of abstraction based on information gleaned from the local database. For instance, in the context of the printed circuit board problem the model at a printed circuit assembly node would take as input a design and associated printed circuit board, and would return manufacturing cost and time. Node-specific models are immobile agents that encapsulate proprietary information that would otherwise be unavailable for general access (see Figure 7.5). The distributed databases and models support information requests, but the network nodes do not permit external agents to reside locally.

Evaluation

Under the constraint that network nodes do not allow external applications to execute locally, it would not be possible to implement the distributed decision framework. Therefore, only the centralized framework would work in this environment. The principal issue that arises then is regarding the local access delays, since network access delays cannot be avoided. Unless local database access delays can be brought down to the order of a few milliseconds per access, this computational environment would not be able to support reasonably efficient evolutionary decision-making. A strategy to improve the local access response time is through database caches that are able to handle complex queries without having to refer to the master database, caches that automatically receive updates from the master database when changes occur, and database servers that exploit parallel processing hardware.[4]

[4]These features are already available in Oracle's Oracle8*i* [ora, 2000a].

8.5.2.2 *Proposal-B*

Assumptions

Each network node has large back-end master database tables but most of the information processing is performed at the level of a sophisticated front-end communicating parallel cache (see Figure 8.17) that executes over a server farm.[5] A server farm allows multiple users to simultaneously and rapidly access the information at a network node without frequent access of the master database tables, which requires slow disk operations. Also, the cache is programmed to receive automatic updates on any relevant state changes to the master database, and it is assumed that the average response times for queries are of the order of a few milliseconds. Importantly, the

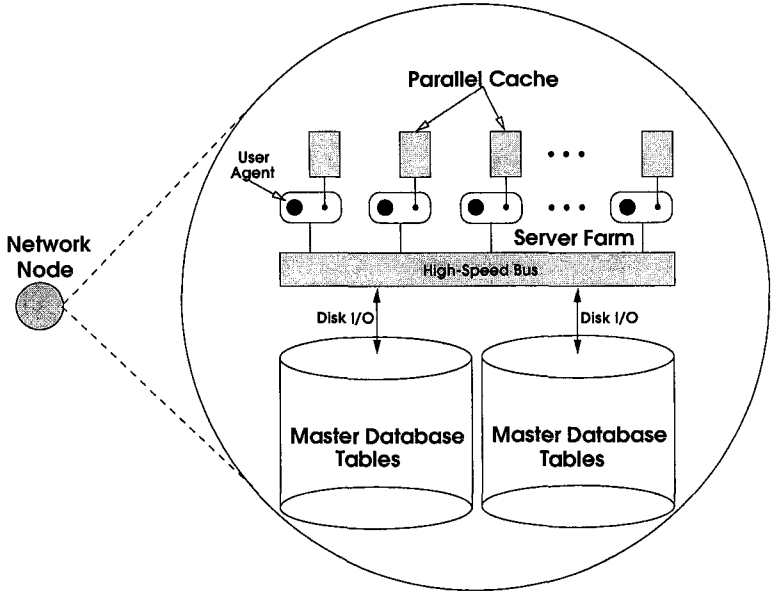


Fig. 8.17 Computational components of a network node that supports fast front-end parallel processing of database accesses.

front-end server farm allows external and local agents to coexist and execute locally.

[5]These features are already supported in Oracle's Oracle Parallel Server [ora, 2000b].

Evaluation

The principal advantages of this environment are speed of local computation and database access, and support of local execution of local and external agents. Database access is rapid on average since the front-end cache simulates a fast local database, and agents are able to benefit from locality of information. Therefore, such a computational environment is highly supportive of efficient network-distributed coevolutionary computation.

In distributed coevolutionary computation, agents migrate between nodes, and when they do they carry with them a small portion of the available local information at a network node to support computations at another network node. While this concept is computationally attractive, and computationally justified since it facilitates the high computational efficiency of the distributed scheme, it does raise issues regarding information confidentiality. A simple solution to this problem is for the organizational entities representing the various network nodes to have prior agreements that support cooperative information sharing. Alternatively, effort could be put in to ensure that the agents that migrate between nodes are engineered to not divulge proprietary information while at the same time supporting cooperative computation across the network nodes.

8.5.2.3 *Discussion*

Both proposals above require investments in speeding-up local computations by means of parallel processing and sophisticated caching of information in large back-end master tables. Therefore, there is no work-around to speeding up local computations at a network node, and this is vital to efficient implementation of the evolutionary decision-making schemes. An interesting observation is that many widely accessed electronic commerce sites today have no choice but to invest in response time enhancement measures in order to be competitive.

The principal difference between the proposals is in their support of local residence of external agents. In Proposal-A, it is assumed that such a feature is not supported, and in this case there is not much of a choice but to use a centralized mode of decision-making. In Proposal-B, it is assumed that the environment supports local residence of external agents, and when such a feature is available, it is computationally very advantageous to consider a distributed mode of decision-making.

This page is intentionally left blank

Chapter 9

Conclusions

This final chapter presents a summary of contributions, potential applications, and directions for future research and development.

9.1 Summary

The fundamental theme of this book is "a systematic development of scalable and efficient evolutionary techniques for network-distributed decision-making." This theme links ideas and contributions that span the following areas:

- Posing the determination of efficient enterprise-level collaborative partnerships as distributed decision problems, and highlighting the importance of software systems that can automate significant portions of these decision tasks. Such decision problems and their solutions methodologies are fundamental in linking to the scientific principles that underlie operations of collaborative enterprise systems.
- Formal modeling and analysis of a class of distributed decision problems arising in integrated design, supplier, and manufacturing planning for modular products, developed as a set of coupled nonlinear assignment problems. This formulation is applicable to a variety of assembly-oriented design-manufacturing domains where integrated design-supplier-manufacturing planning decisions are desired.
- An analysis of the optimization algorithms that work well for this class of nonlinear assignment problems, selection of evolutionary techniques for this problem class, and development of a theoretical foundation to explain evolutionary algorithm behavior through proofs of convergence and convergence rates.

- Development of a novel coevolutionary algorithm based on distributed evolutionary algorithm components and software agents that facilitate network-efficient cooperative exploration of highly coupled discrete spaces arising in distributed decision problems.
- Development of a theoretical foundation to explain coevolutionary algorithm behavior through proofs of convergence and convergence rates. Importantly, this work helps clarify the conditions for global convergence of coevolutionary algorithms.
- Development and analysis of techniques for achieving low convergence error and high computational efficiency with distributed coevolutionary algorithms, and an investigation of the factors that influence distributed convergence.
- Development of prototype CVDE implementations that utilize distributed evolutionary agents and mobile agents as principal entities that generate and execute queries and transactions among distributed applications and databases to support a global optimization of design, supplier, and manufacturing planning for printed circuit assemblies.
- An evaluation of expected performance of the decision-making frameworks over the general internet for realistic problem sizes, and an analysis of the computational requirements for implementing these frameworks to support network-enabled design and manufacturing organizations.

9.2 Future Work

There are several interesting opportunities for further research and development based on the work in this book. While some of these are immediate extensions to this work, others are motivated by a broad vision for the future on the diverse applicability of efficient network-distributed decision-making systems.

9.2.1 *Multi-Criteria Optimization*

For the design-supplier-manufacturing planning problem class developed in Chapter 3, the objective is an aggregate trade-off function of cost and time factors, which themselves are global functions of the various assignments. In several situations there is an advantage to treating the cost and time factors without aggregation. In these cases, evolutionary multi-criteria

optimization methods assume importance, and continuing work (see for example [Bonissone and Subbu, 2002; Graves *et al.*, 2003]) focuses on these aspects.

Methods for multi-criteria optimization and decision-making constitute a critical focus area for ongoing research and development activities at General Electric Research and is showing significant promise across several complex business decision-making and engineering design problems [Tappeta *et al.*, 2002]. Evolutionary multi-criteria optimization algorithms recently developed by the first author are being utilized by General Electric's financial systems business for making multi-billion dollar investment decisions. This research is soon to be published. In this decision-making problem the evolutionary search is based on multiple linear and nonlinear measures of return and risk, and it is particularly valuable to identify the space of Pareto-optimal or efficient investment decisions.

9.2.2 *Domain Heuristics*

Recent insight due to the *No Free Lunch* theorem suggests that any given algorithm cannot be expected to produce consistently superior results over the space of optimization problems [Wolpert and MacReady, 1997]. One interpretation of this theoretical result discourages the search for the best algorithm since all algorithms on average behave the same across the space of all problems. However, the alternative positive and more useful interpretation of this result is that it is impossible to find a unique best algorithm for all problems, therefore, it is important to enhance the search performance of a given algorithm with whatever domain knowledge is available to solve that problem. This latter interpretation via the use of domain heuristics and evolutionary computation has proved highly successful for several complex business and engineering problems at General Electric Research [Bonissone *et al.*, 2002; Bush and Kulkarni, 2002]. Leveraging domain knowledge is a critical component of the problem solving process and has the power to equip evolutionary and coevolutionary search methods with the power to tackle complex optimization problems.

9.2.3 *Distributed Convergence*

This book develops a theoretical foundation for modeling and convergence analysis of centralized and distributed coevolutionary algorithms, and con-

vergence and convergence rate results are derived for basic objective function classes based on Gaussians. While the method of "Sum of Gaussians" is fairly general and can be used to represent a variety of objective functions, it is attractive to investigate the scope of derivation of convergence rate results for more complex landscapes.

In large heterogeneous information networks there is the real possibility that there will be significant disparities in computational resources at various network nodes, and the edges interconnecting these widely distributed nodes may also have significant disparities in access delays. The distributed coevolutionary algorithms developed in this book function synchronously, and there is the potential that the slowest node/edge may significantly affect the network-based performance of the entire system. Thus, there is a need to develop more extensive methods based on simulation and analysis to evaluate the combined performance and reliability of these algorithms in large, highly heterogeneous information networks. Discrete-event systems modeling tools such as the Petri-Net model are advantageous for this level of modeling and analysis [Desrochers *et al.*, 2003].

9.2.4 *Robust Optimization*

Traditionally, much emphasis is placed on identifying a globally optimal solution. However, a solution that is globally optimal but is highly sensitive to small changes in decision variables would in practice be less attractive than a solution that is good and is stable over small changes in decision variables. There is an advantage therefore to develop techniques that are able to identify solutions that are less sensitive to uncertainties in decision variables.

Robustness is a concept that is well developed in the control systems literature, and is used in the context of design of controllers that are relatively insensitive to variations in parameters that determine plant performance [Ackermann, 1991]. In these systems the plant model is assumed to be not known exactly and the main concern is to design a control scheme that satisfies performance specifications for all plants in the uncertainty set [Milanese *et al.*, 1991]. Recently, robustness has been discussed in the context of mathematical optimization (e.g. [Mulvey *et al.*, 1995; Bai *et al.*, 1997]). The fundamental goal of robust optimization is to identify an optimal or near-optimal solution that is not very sensitive to uncertainty in decision-variables. Robust optimization explicitly incorporates robustness measures in the problem formulation in order to identify solutions that

are less sensitive to uncertainties in decision-variables.

9.2.5 *Prototype and Model Development*

There is a strong need to adapt the prototype CVDE described in this book to function in commercial applications settings that require access to commercial databases. Linked to this is a need for development or acquisition of more realistic and complete domain-specific models. Such advanced implementations would require a significant emphasis on developing graphical user interfaces and displays that help users participate in the decision-making process, and multi-criteria decision-making methods that include human decision makers [Josephson *et al.*, 1998] will especially be critical to the success of such systems. Ongoing work at General Electric Research focuses on the development of such interactive multi-criteria decision-making tools for complex problems [Cheetham, 2003].

9.2.6 *Applications*

The methodology presented in this book can have a fundamental impact on the principles and practice of engineering in industrial product development in the network-based distributed environment that is emerging within and among corporate enterprise systems. In addition, the conceptual framework of the approach to distributed decision systems presented in this book may have much wider implications for network-based systems ranging from intelligent agent-based browser systems, to enhanced consumer and business services, and intelligent search techniques in scientific and commercial databases.

The following discussion presents two application examples; one from integrated circuit design and manufacturing planning, and the second from decentralized air traffic management.

9.2.6.1 *IC Design and Manufacturing*

Integrated circuit (IC) design and manufacturing, is an area deemed critical to the sustained growth of an economy driven by advances in information technology and one consuming considerable resources.

The largest delay in bringing new ICs to market has been, and will continue to be for the foreseeable future, the design and verification process. This is particularly true of mixed-signal designs, Systems on a Chip (SOC) implementations and densely packed Application Specific Integrated Cir-

cuits (ASIC). Design reuse of complex macro-cells, also called Intellectual Property (IP) cores, is a growing response to the design and verification process in which previously developed and proven sub-chip modules are reused in new ICs. This concept extends the well-developed use of in-house design libraries to include complex macro-cells as basic building blocks, similar to the use of integrated circuits as basic building blocks for electronics products. This is a significant advantage both for the IC designer, and with widespread implementation, the industry as well [Gutmann *et al.*, 1999; Gutmann, 1999].

In support of this trend, several commercial IC foundries have helped reduce the technological gap enjoyed by vertically integrated industrial leaders, and as a consequence, most IC designers have access to near state-of-the-art fabrication facilities. Licensing of fully verified complex macro-cells can thus reduce the time-to-market appreciably, and simultaneously reduce new product risk. The advantage of such a design approach will lead to more standardized design and manufacturing processes among design groups and leading-edge IC foundries, and offset the IC designer mind-set of designing from scratch, which is a principal factor causing design delays. The time-to-market advantage of reusing IP cores will therefore be overwhelming.

In the anticipated design and manufacturing environment, IC *virtual design* becomes analogous to virtual design in the printed circuit assembly domain, where selection of IP cores is analogous to selection of packaged ICs, and selection of foundries is analogous to selection of assembly facilities. In this environment, most of the IC design effort would focus on the development of interconnects and "glue" circuits, while benefiting significantly from the reuse of complicated but fully verified basic building blocks. A common unifying theme in these applications domains is that multiple logically interrelated decision resources (IP cores, their suppliers, and foundries) are physically distributed, and information for global decision-making using these resources is available from several network-distributed databases.

9.2.6.2 *Decentralized Air Traffic Management*

The current air traffic control system in place in the United States, Europe and other countries is based on a network of geographically contiguous sectors, one or more of which are intersected by en-route flights between any two points. Each sector may be further decomposed into a set of

smaller control air-spaces. Each such airspace is typically managed by an expert air traffic controller who monitors the numbers and trajectories of the various flights in their given airspace and coordinates the flights in their airspace in collaboration with neighboring controllers. Since controllers wish to retain as much control over flight paths as possible, aircraft are currently disallowed to pick arbitrary routes and are assigned to one of a few standard routes between an origin and destination. While this system has been in place for decades and is considered reliable and robust, the increasing demand for air travel is frequently causing the system to reach an overloaded congested state at several sectors. Congestion occurs when the number of active flights in a controller's airspace exceeds the number of flights that they can safely manage.

Current efforts at General Electric Research to alleviate this congestion problem include the development of methods to better predict congestion using temporal reasoning tools, and global optimization to realize optimal flight routes. Since a flight between an origin and destination is typically restricted to a few standard routes, the assignment of flights to routes such that the expected congestion is minimized is a discrete planning problem not unlike the class of problems discussed in this book. In fact, a suitable representation for solving this problem is similar to the representation for the printed circuit assembly planning problem discussed in this book. From a practical perspective it would be impossible to simultaneously optimize the trajectories of all flights in a given time window over the entire national airspace. In addition, since a given sector is tightly coupled to only a few other sectors in a time window due to shared intersecting flights, flight planning and optimization for the entire national airspace for given time window horizons is more efficiently performed in a decentralized manner. The algorithms and architectures presented in this book may be readily applicable in this environment.

This page is intentionally left blank

Appendix A

Evolutionary Algorithm Theory

This chapter presents two key derivations that support the development of evolutionary algorithm theory.

A.1 Population Distribution Evolution

The objective function is $\psi(x) = \mathsf{N}[x^*, K]$, and the initial distribution $P_0(dx) = \mathsf{N}[x_0, C_0]\mu_n(dx)$. The product $P_0(dx)\psi(x)$ is given by the scaled Gaussian

$$\frac{1}{(2\pi)^{\frac{n}{2}}|KC_0K^{-1} + K|^{\frac{1}{2}}} \exp\left(-\frac{1}{2}(x^* - x_0)^T(K + C_0)^{-1}(x^* - x_0)\right) \quad \text{(A.1)}$$
$$\mathsf{N}[(K^{-1} + C_0^{-1})^{-1}(K^{-1}x^* + C_0^{-1}x_0), (K^{-1} + C_0^{-1})^{-1}]\mu_n(dx)$$

The first term from above,

$$\frac{1}{(2\pi)^{\frac{n}{2}}|KC_0K^{-1} + K|^{\frac{1}{2}}} \exp\left(-\frac{1}{2}(x^* - x_0)^T(K + C_0)^{-1}(x^* - x_0)\right) \quad \text{(A.2)}$$

is a scalar. Therefore the integral

$$\int_{\mathcal{X}} P_0(dx)\psi(x) \quad \text{(A.3)}$$

is simply the above scalar since the Gaussian over the space $\mathcal{X}$ integrates to unity. Since the stochastic variation at any generation consists of an n-dimensional, zero-mean, coordinate-wise independent Gaussian, the integral

$$\int_{\mathcal{X}} P_0(dz)\psi(z)Q_0(z, dx) \quad \text{(A.4)}$$

is a simply a convolution of the scaled Gaussian $P_0(dz)\psi(z)$ and $Q_0(z,\cdot)$, which results in a Gaussian with no change in its mean and whose covariance is the sum of the covariances of $P_0(dz)\psi(z)$ and $Q_0(z,\cdot)$. Therefore

$$P_1(dx) = \frac{\int_{\mathcal{X}} P_0(dz)\psi(z)Q_0(z,dx)}{\int_{\mathcal{X}} P_0(dz)\psi(z)} \tag{A.5}$$

is simply $\mathsf{N}[(K^{-1}+C_0^{-1})^{-1}(K^{-1}x^*+C_0^{-1}x_0), W_0+(K^{-1}+C_0^{-1})^{-1}]\mu_n(dx)$. By induction, $P_{g+1}(dx) = \mathsf{N}[(K^{-1}+C_g^{-1})^{-1}(K^{-1}x^*+C_g^{-1}x_g), W_g+(K^{-1}+C_g^{-1})^{-1}]\mu_n(dx)$.

A.2 Proof of Positive Definiteness

$[I-(I+KC_g^{-1})^{-1}]$ can be rewritten as $(I+C_gK^{-1})^{-1}$. For convenience, A_g is written as $A_g = (I+C_gK^{-1})^{-1}$. Then $A_0 = (I+C_0K^{-1})^{-1}$, and since $C_0 = cI, c>0$, $A_0 = (I+cK^{-1})^{-1}$. Since K is positive definite, A_0 is positive definite. $A_1 = (I+C_1K^{-1})^{-1}$, and $C_1 = \sigma_0^2 I + (K^{-1}+C_0^{-1})^{-1}$. Substituting C_1 in A_1 gives $A_1 = [I+(\sigma_0^2 I + (K^{-1}+\frac{1}{c}I)^{-1})K^{-1}]^{-1}$, that reduces to $A_1 = [I+\sigma_0^2K^{-1}+(I+\frac{1}{c}K)^{-1}]^{-1}$, which is clearly positive definite. Continuing in this fashion, and by induction, A_g is always positive definite.

Appendix B

Models for the Printed Circuit Assembly Problem

This chapter presents internals of several heuristically-based models used in the printed circuit assembly planning problem.

B.1 Part

Table B.1 shows the principal characteristics of each available part from a parts distributor. The package type identifiers used are: surface-mount (SM), through-hole (TH). The package size identifiers used are: very small (VS), small (S), medium (M), large (L), very large (VL).

Table B.1 Principal characteristics of an available part.

Characteristic	Description
`identity`	A unique identifier for a part.
`moduleType`	An identifier for the type of module the part belongs to.
`cost`	A non-negative real number.
`leadTime`	A non-negative real number.
`packageType`	An identifier from the set [SM, TH].
`packageSize`	An identifier from the set [VS, S, M, L, VL].

B.2 Design

Table B.2 shows the principal characteristics of a design entity, which is an aggregate of its parts. All design characteristics above except `componentArea` are explained therein, and now the computation of `componentArea` is described. Package sizes of parts are coded as follows: VS = 1, S = 5, M = 10, L = 20, VL = 50. The normalized component area

Table B.2 Principal characteristics of a design.

Characteristic	Description
`numParts`	Number of parts in the design.
`netCost`	Net cost of parts in the design.
`maxLeadTime`	Maximum lead time of parts in the design.
`componentArea`	Normalized area occupied by parts in the design.
`percentSM`	Percentage of surface-mount components in the design.
`numSM`	Number of surface-mount components in the design.
`numFineSM`	Number of surface-mount components with package size VS.
`numRegularSM`	Number of surface-mount components with package sizes [S, M, L, VL].
`numTH`	Number of through-hole components in the design.
`numRegularTH`	Number of through-hole components with package sizes [S, M].
`numLargeTH`	Number of through-hole components with package sizes [L, VL].

of a part is the ratio of its package size and the largest package size, so the component area of parts in a design is the sum of the individual normalized component areas.

B.3 Printed Circuit Board

A printed circuit board is generated by a heuristically-based printed circuit board generator that takes as input a design, and outputs printed circuit board characteristics. The annotated internals of the printed circuit board generator are shown below.

```
// Assume that the largest board desirable is one whose
// componentArea = 20
maxBoardArea = 20;

// Compute the board area. Assume that the area of the
// board is 1.5 the area occupied by the components. However,
// assume that as the percentage of surface mounts increases
// the board area decreases by upto two thirds.
boardArea = (1.5-design.percentSM)*design.componentArea;

// Compute the size ratio of the board. If the generated
// board is larger than the largest board desirable, and
// if the percentage of SM components is beyond a threshold
```

```
// make a two-sided board of half the size.
sizeFactor = boardArea/maxBoardArea;
numSides = 1;
if((sizeFactor > 1.0) && (design.percentSM >= 0.3)) {
  sizeFactor /= 2;
  numSides = 2;
}

// Compute the layer density of the board. Assume that as
// the percentage of SM components increases, the layer
// density increases. Also, as the board grows in size there
// is less of a need for multiple layers.
layerDensity = (numSides-1)*pow(design.percentSM,2)/exp(sizeFactor);

// Compute the SM pad density. Assume that as the percentage
// of surface mount components increases, the pad density
// increases quickly. Simultaneously, as the board size
// increases the SM pad density decreases.
SMPadDensity = (1-exp(-design.percentSM*4)+exp(-4))/exp(sizeFactor);

// Compute the hole density. Assume that the number of holes
// increases when the percentage of through hole components
// increases. Simultaneously, as the board size increases the
// hole density decreases.
holeDensity = (1-exp(-(1-design.percentSM)*9)+exp(-9))/
   exp(sizeFactor);
```

B.4 Printed Circuit Board Fabrication Line

Table B.3 shows the principal characteristics of a printed circuit board fabrication line. For printed circuit board fabrication lines, it is assumed

Table B.3 Principal characteristics of a printed circuit board fabrication line.

Characteristic	Description
`identity`	A unique identifier for a fabrication line.
`lineOverhead`	A real number index in the range $[0, 1]$.
`lineEfficiency`	A real number index in the range $[0, 1]$.

that the principal differentiators between lines are overhead and efficiency.

While overhead is directly proportional to cost of fabrication, efficiency is inversely proportional to fabrication time.

A printed circuit board fabrication line takes a design and its associated printed circuit board as inputs, and outputs heuristically computed cost and time for fabrication.

Fabrication Cost

```
fabricationCost  = pcboard.sizeFactor +
                   exp(pcboard.layerDensity) +
                   design.percentSM * pcboard.SMPadDensity *
                   pcboard.numSides + pcboard.holeDensity;
fabricationCost *= (1+lineOverhead);
```

Fabrication Time

```
fabricationTime  = pcboard.sizeFactor +
                   exp(pcboard.layerDensity) +
                   design.percentSM * pcboard.SMPadDensity *
                   pcboard.numSides + pcboard.holeDensity;
fabricationTime /= (1+lineEfficiency);
```

B.5 Printed Circuit Assembly Line

Table B.4 shows the principal characteristics of the printed circuit assembly line. At the abstraction level of a printed circuit assembly line the principal

Table B.4 Principal characteristics of a printed circuit assembly line.

Characteristic	Description
`identity`	A unique identifier for an assembly line.
`lineOverhead`	A real number index in the range $[0, 1]$.
`lineEfficiency`	A real number index in the range $[0, 1]$.
`cellList`	A list of manufacturing cells that constitute the line.

differentiators between lines are overhead, efficiency, and characteristics of the manufacturing cells that constitute a line.

Each manufacturing line is a sequence of several manufacturing cells, and each cell realizes a subtask in the overall printed circuit assembly process. Table B.5 shows the principal characteristics of a printed circuit as-

sembly cell. A cell of the surface-mount (SM) type handles assembly of all surface-mount components with package sizes [S, M, L, VL], a cell of the fine surface-mount (FSM) type handles assembly of very small surface-mount components, a cell of the through-hole (TH) type performs placement of through-hole components with package sizes [S, M] (it is assumed that through-hole components cannot have a package size of VS), a cell of the large through-hole (LTH) type performs placement of through-hole components with package sizes [L, VL], a cell of the wave solder (WS) type solders all through-hole components, and a cell of the final assembly (FIN) type performs cleaning and inspection of the assembled product.

Table B.5 Principal characteristics of a printed circuit assembly cell.

Characteristic	Description
`identity`	A unique identifier for an assembly cell.
`type`	An identifier from the set [SM, FSM, TH, LTH, WS, FIN].
`avgTimePerPart`	A non-negative real number that represents the average time required to handle a part.
`avgTimePerBoard`	A non-negative real number that represents the average time required to handle a board.
`costPerUnitTime`	A non-negative real number that represents the cost per unit time of cell use.

A printed circuit assembly line takes a design and its associated printed circuit board as inputs, and outputs heuristically computed cost and time for manufacturing. The inputs are processed by each of the constituent cells in a manufacturing line.

Manufacturing Cost

For assembly cells of type [SM, FSM, TH, LTH, WS], the manufacturing cost incurred by a cell, `cellManufacturingCost`, is computed as

```
cellManufacturingCost  = n * avgTimePerPart * costPerUnitTime;
cellManufacturingCost += avgTimePerBoard * costPerUnitTime;
```

where `n` is the number of the parts with the specific types and sizes that the cell is equipped to handle. For assembly cell of type FIN, the `cellManufacturingCost` is computed as

```
cellManufacturingCost = avgTimePerBoard * costPerUnitTime;
```

$$\texttt{manufacturingCost} = \sum_{\texttt{cellList}} \texttt{cellManufacturingCost}$$

```
manufacturingCost *= (1+pcboard.sizeFactor);
manufacturingCost *= (1+lineOverhead);
```

Manufacturing Time

For assembly cells of type [SM, FSM, TH, LTH, WS], the manufacturing time incurred by a cell, `cellManufacturingTime`, is computed as

```
cellManufacturingTime  = n * avgTimePerPart;
cellManufacturingTime += avgTimePerBoard;
```

where `n` is the number of the parts with the specific types and sizes that the cell is equipped to handle. For assembly cell of type FIN, the `cellManufacturingTime` is computed as

```
cellManufacturingTime = avgTimePerBoard;
```

$$\texttt{manufacturingTime} = \sum_{\texttt{cellList}} \texttt{cellManufacturingTime}$$

```
manufacturingTime *= (1+pcboard.sizeFactor);
manufacturingTime /= (1+lineEfficiency);
```

Bibliography

(1991a). *21st Century Manufacturing Enterprise Strategy: An Industry-Led View.* Iacocca Institute, Lehigh University. Volume 1.

(1991b). *21st Century Manufacturing Enterprise Strategy: Infrastructure.* Iacocca Institute, Lehigh University. Volume 2.

(2000a). *Oracle8i.* Oracle Corp., http://www.oracle.com.

(2000b). *Oracle8i Parallel Server.* Oracle Corp., http://www.oracle.com.

(2000). *Voyager.* ObjectSpace Inc., http://www.objectspace.com.

Aarts, E. H. L., Korst, J. H. M., and van Laarhoven, P. J. M. (1997). Simulated annealing. In Aarts, E. and Lenstra, J. K., editors, *Local Search in Combinatorial Optimization.* John Wiley.

Ackermann, J. (1991). Robust car steering by yaw rate feedback. In Bhattacharyya, S. P. and Keel, L. H., editors, *Control of Uncertain Dynamic Systems.* CRC Press.

Bäck, T. (1995). Order statistics for convergence velocity analysis of simplified evolutionary algorithms. In *Proceedings of the Foundations of Genetic Algorithms.* Morgan Kaufman.

Bäck, T. (1996). *Evolutionary Algorithms in Theory and Practice.* Oxford University Press, New York.

Bäck, T. (1997). Heuristics for parameter-setting issues: Mutation parameters. In Bäck, T., Fogel, D., and Michalewicz, Z., editors, *Handbook of Evolutionary Computation.* Oxford University Press.

Bäck, T. (1998). On the behavior of evolutionary algorithms in dynamic environments. In *Proceedings of the World Conference on Computational Intelligence*, Anchorage, Alaska.

Bäck, T., Heistermann, J., Kappler, C., and Zamparelli, M. (1996). Evolutionary algorithms support refueling of pressurized water reactors. In *Proceedings of the IEEE International Conference on Evolutionary Computation*, Nagoya, Japan.

Bäck, T. and Schwefel, H.-P. (1996). Evolutionary computation: An overview. In *Proceedings of the IEEE International Conference on Evolutionary Computation*, Nagoya, Japan.

Bai, D., Carpenter, T., and Mulvey, J. (1997). Making a case for robust opti-

mization models. *Management Science*, 43(7).

Barnier, N. and Brisset, P. (1998). Optimization by hybridization of a genetic algorithm with constraint satisfaction techniques. In *Proceedings of the World Conference on Computational Intelligence*, Anchorage, Alaska.

Bellman, R. and Dreyfus, S. (1962). *Applied Dynamic Programming.* Princeton University Press, New Jersey.

Bertsekas, D. P. and Tsitsiklis, J. N. (1997). *Parallel and Distributed Computation: Numerical Methods.* Athena Scientific, Belmont, Massachusetts.

Blickle, T. and Thiele, L. (1997). A comparison of selection schemes used in evolutionary algorithms. *Evolutionary Computation*, 4(4).

Bonissone, P. P., Subbu, R., and Aggour, K. (2002). Evolutionary optimization of fuzzy decision systems for automated insurance underwriting. In *Proceedings of the World Conference on Computational Intelligence*, Honolulu, Hawaii.

Bonissone, S. and Subbu, R. (2002). Exploring the pareto frontier using multisexual evolutionary algorithms: an application to a flexible manufacturing problem. In *Proceedings of the SPIE Annual Meeting-Program on Algorithms and Architectures*, Seattle, Washington.

Booch, G. (1994). *Object-Oriented Analysis and Design.* Benjamin Cummings, California.

Boothroyd, G. (1994). Product design for manufacture and assembly. *Computer-Aided Design*, 26(7).

Boothroyd, G. and Dewhurst, P. (1983). *Product Design for Assembly.* Boothroyd Dewhurst, Inc., Wakefield, Rhode Island.

Bürger, R. (1988). Mutation-selection balance and continuum-of-alleles models. *Mathematical Biosciences*, 91.

Burke, E. K., Newall, J. P., and Weare, R. F. (1998). Initialization strategies and diversity in evolutionary timetabling. *Evolutionary Computation*, 6(1).

Bush, S. F. and Kulkarni, A. B. (2002). Genetically induced communication network fault tolerance. In *Proceedings of the Santa Fe Institute Workshop: Resilient and Adaptive Defence of Computing Networks*, Santa Fe, New Mexico.

Cantú-Paz, E. (1999). *Designing Efficient and Accurate Parallel Genetic Algorithms.* PhD thesis, University of Illinois at Urbana-Champaign, Urbana, Illinois.

Capcarrère, M., Tomassini, M., Tettamanzi, A., and Sipper, M. (1999). A statistical study of a class of cellular evolutionary algorithms. *Evolutionary Computation*, 7(3).

Cedeño, W., Vemuri, V. R., and Slezak, T. (1995). Multiniche crowding in genetic algorithms and its application to the assembly of DNA restriction-fragments. *Evolutionary Computation*, 2(4).

Chakraborty, U. K., Deb, K., and Chakraborty, M. (1997). Analysis of selection algorithms: A markov chain approach. *Evolutionary Computation*, 4(2).

Chandrasekaran, B. (1981). Natural and social system metaphors for distributed problem solving: Introduction to the issue. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-11(1).

Chao, K.-M., Anane, R., Norman, P., Tasi, C.-F., and Reeves, C. (2002). Multiple evolution and concurrent engineering design. In *The 7th International Conference on Computer Supported Cooperative Work in Design.*

Cheetham, W. (2003). Global grade selector: A recommender system for supporting the sale of plastic resin. In *Proceedings of the 5th International Conference on Case-Based Reasoning*, Trondheim, Norway.

Chen, R. (1996). A benchmark comparing Unix and Windows NT for decision support with Oracle RDBMS. In *CMG Proceedings*, volume 1.

Cheng, T. C. E. and Sin, C. C. S. (1990). A state-of-the-art review of parallel-machine scheduling research. *European Journal of Operational Research*, 47(3).

Chu, P. C. and Beasley, J. E. (1997). A genetic algorithm for the generalized assignment problem. *Computers and Operations Research*, 24(1).

Davis, T. E. and Principe, J. C. (1993). A markov chain framework for the simple genetic algorithm. *Evolutionary Computation*, 1(3).

De Jong, K. A. (1975). *An Analysis of the Behavior of a Class of Genetic Adaptive Systems.* PhD thesis, University of Michigan, Ann Arbor, Michigan.

De Jong, K. A., Spears, W. M., and Gordon, D. F. (1994). Using markov chains to analyze GAFOs. In *Proceedings of the Foundations of Genetic Algorithms.*

Deb, K. and Chakroborty, P. (1998). Time scheduling of transit systems with transfer considerations using genetic algorithms. *Evolutionary Computation*, 6(1).

Desrochers, A., Graves, R. J., Sanderson, A. C., Goel, A., Scales, R., Xu, T., Xue, F., Bonissone, S., and Subbu, R. (2003). Advanced research in scalable enterprise systems: Network-based distributed relational decision framework for scalable enterprise systems-phase II. In *Proceedings of the 2003 NSF Design, Service and Manufacturing Grantee's Conference*, Birminghan, Alabama.

Dimopoulos, C. and Zalzala, A. M. S. (2000). Recent developments in evolutionary computation for manufacturing optimization: Problems, solutions, and comparisons. *IEEE Transactions on Evolutionary Computation*, 4(2).

Dowlatshahi, S. (1992). Product design in a concurrent engineering environment: an optimization approach. *International Journal of Production Research*, 30(8).

Ficici, S. G., Melnick, O., and Pollack, J. B. (2000). A game-theoretic investigation of selection methods used in evolutionary algorithms. In *Proceedings of the IEEE Congress on Evolutionary Computation*, San Diego, California.

Fogel, D. B. (1995). *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence.* IEEE Press, New Jersey.

Fogel, L. J., Owens, A. J., and Walsh, M. J. (1966). *Artificial Intelligence Through Simulated Evolution.* John Wiley, New York.

Fonseca, C. M. and Fleming, P. J. (1998a). Multiobjective optimization and multiple constraint handling with evolutionary algorithms–Part I: A unified formulation. *IEEE Transactions on Systems, Man, and Cybernetics–Part A*, 28(1). Systems and Humans.

Fonseca, C. M. and Fleming, P. J. (1998b). Multiobjective optimization and mul-

tiple constraint handling with evolutionary algorithms–Part II: Application example. *IEEE Transactions on Systems, Man, and Cybernetics–Part A*, 28(1). Systems and Humans.

Gao, Y. (1998). Comments on theoretical analysis of evolutionary algorithms with an infinite population size in continuous space, Part-I: Basic properties of selection and mutation. *IEEE Transactions on Neural Networks*, 9(2).

Glover, F. (1986). Future paths for integer programming and links to artificial intelligence. *Computers and Operations Research*, 13(5).

Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison-Wesley, Massachusetts.

Goldfarb, D. and Todd, M. J. (1989). Linear programming. In Nemhauser, G. L., Kan, A. H. G. R., and Todd, M. J., editors, *Optimization*, volume 1. North-Holland.

Gondran, M. and Minoux, M. (1984). *Graphs and Algorithms.* John Wiley, New York.

Graves, R. J., Sanderson, A. C., Xue, F., Bonissone, S., and Subbu, R. (2003). Advanced research in scalable enterprise systems: Multi-objective optimization. In *Proceedings of the 2003 International Conference on Industrial Engineering and Production Management*, Porto, Portugal.

Gray, J. (1988). The cost of messages. In *Proceedings of the Seventh Annual ACM Symposium on Principles of Distributed Computing*, Toronto, Canada.

Grefenstette, J. J. (1986). Optimization of control parameters for genetic algorithms. *IEEE Transactions on Systems, Man, and Cybernetics*, 16(1).

Grefenstette, J. J. (1992). Genetic algorithms for changing environments. In *Proceedings of Parallel Problem Solving From Nature-2.* North-Holland.

Grefenstette, J. J. (1995). Predictive models using fitness distributions of genetic operators. In *Proceedings of the Foundations of Genetic Algorithms.* Morgan Kaufman.

Grefenstette, J. J. (1999). Evolvability in dynamic fitness landscapes: A genetic algorithm approach. In *Proceedings of the IEEE Congress on Evolutionary Computation*, Washington D. C.

Gutmann, R. J. (1999). Advanced silicon IC interconnect technology and design: Present trends and RF wireless implications. *IEEE Transactions on Microwave Theory and Techniques*, 47(6).

Gutmann, R. J., Chan, K., and Graves, R. J. (1999). Interconnect technology and design implications for future ASIC and System-on-a-Chip (SOC) implementations. In *Proceedings of the Advanced Semiconductor Manufacturing Conference and Workshop*, Boston, Massachusetts.

Hart, E., Ross, P., and Nelson, J. (1998). Solving a real-world problem using an evolving heuristically driven schedule builder. *Evolutionary Computation*, 6(1).

Hayes, C. C. (1999). Agents in a nutshell–a very brief introduction. *IEEE Transactions on Knowledge and Data Engineering*, 11(1).

He, D. W. and Kusiak, A. (1997). Design of assembly systems for modular products. *IEEE Transactions on Robotics and Automation*, 13(5).

Hertz, A., Taillard, E., and de Werra, D. (1997). Tabu search. In Aarts, E. and

Lenstra, J. K., editors, *Local Search in Combinatorial Optimization.* John Wiley.

Hocaoğlu, C. (1997). *Multipath Planning using Evolutionary Computation with Speciation.* PhD thesis, Rensselaer Polytechnic Institute, Troy, NY.

Hocaoğlu, C. and Sanderson, A. C. (1996). Implementation and assesment of a distributed, object-oriented information infrastructure for agile electronics manufacturing. In *Proceedings of the Third ISPE International Conference on Concurrent Engineering*, Toronto, Canada.

Hohenstein, U., Pleßer, V., and Heller, R. (1997). Evaluating the performance of object-oriented database systems by means of a concrete application. In *Database and Expert Systems Applications Proceedings.*

Holland, J. H. (1994). *Adaptation in Natural and Artificial Systems: an introductory analysis with applications to biology, control, and artificial intelligence.* The MIT Press, Cambridge, Massachusetts, third edition. First Edition: 1975 The University of Michigan.

Hsieh, F.-S. (2002). Design of evolvable manufacturing processes. In *Proceedings of the IEEE Congress on Evolutionary Computation*, Honolulu, Hawaii.

Husbands, P., McIlhagga, M., and Ives, R. (1997). Experiments with an ecosystems model for integrated production planning. In Bäck, T., Fogel, D., and Michalewicz, Z., editors, *Handbook of Evolutionary Computation.* Oxford University Press.

Jennings, N. R., Sycara, K., and Woolridge, M. (1998). A roadmap of agent research and development. *Autonomous Agents and Multi-Agent Systems*, 1(1).

Jennings, N. R. and Woolridge, M. (1998). Applications of intelligent agents. In Jennings, N. R. and Woolridge, M., editors, *Agent Technology: Foundations, Applications, and Markets.* Springer-Verlag.

Joines, J. A., Culbreth, C. T., and King, R. E. (1996). Manufacturing cell design: an integer programming model employing genetic algorithms. *IIE Transactions*, 28(1).

Josephson, J. R., Chandrasekaran, B., Carroll, M., Iyer, N., Wasacz, B., Rizzoni, G., Li, Q., and Erb, D. A. (1998). An architecture for exploring large design spaces. In *Proceedings of the American Association for Artificial Intelligence (AAAI).*

Karlin, S. (1979). Models of multifactorial inheritance: I, multivariate formulations and basic convergence results. *Theoretical Population Biology*, 15.

Kauffman, S. A. and Johnsen, S. (1991). Co-evolution to the edge of chaos: Coupled fitness landscapes, poised states, and co-evolutionary avalanches. *Journal of Theoretical Biology*, 149.

Kernighan, B. W. and Lin, S. (February, 1970). An efficient heuristic procedure for partitioning graphs. *The Bell System Technical Journal*, 49.

Kirkpatrick, S., Jr., C. D. G., and Vecchi, M. P. (1983). Optimization by simulated annealing. *Science*, 220(4598).

Koza, J. (1992). *Genetic programming: On the programming of computers by means of natural selection.* The MIT Press, Cambridge, Massachusetts.

Lee, E. K. and Mitchell, J. E. (1999). Branch-and-bound methods for integer

programming. In *Encyclopedia of Optimization.* Kluwer Academic.

Lesser, V. R. (1991). A retrospective view of FA/C distributed problem solving. *IEEE Transactions on Systems, Man, and Cybernetics*, 21(6).

Lesser, V. R. (1999). Cooperative multiagent systems: A personal view of the state of the art. *IEEE Transactions on Knowledge and Data Engineering*, 11(1).

Lesser, V. R. and Corkill, D. D. (1981). Functionally accurate, cooperative distributed systems. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-11(1).

Liles, W. and De Jong, K. (1999). The usefulness of tag bits in changing environments. In *Proceedings of the IEEE Congress on Evolutionary Computation*, Washington D. C.

Lin, S. and Kernighan, B. W. (1973). An effective heuristic algorithm for the traveling salesman problem. *Operations Research*, 21(2).

Luenberger, D. G. (1984). *Linear and Nonlinear Programming.* Addison-Wesley, Massachusetts.

Maes, P. (1994). Modeling adaptive autonomous agents. *Artificial Life*, 1(2).

Matsumura, T., Nakamura, M., Okech, J., and Onaga, K. (1998). A parallel and distributed genetic algorithm on loosely-coupled multiprocessor systems. *IEICE Transactions on Fundamentals of Electronics, Communications, and Computer Sciences*, E81-A(4).

Michalewicz, Z. (1996). *Genetic Algorithms + Data Structures = Evolution Programs.* Springer-Verlag, New York, Berlin, third edition.

Michalewicz, Z. and Schoenauer, M. (1996). Evolutionary algorithms for constrained parameter optimization problems. *Evolutionary Computation*, 4(1).

Milanese, M., Fiorio, G., Malan, S., and Vicino, A. (1991). Robust performance design of nonlinearily peturbed control systems. In Bhattacharyya, S. P. and Keel, L. H., editors, *Control of Uncertain Dynamic Systems.* CRC Press.

Mitchell, J. E. (1999a). Branch-and-cut algorithms for integer programming. In *Encyclopedia of Optimization.* Kluwer Academic.

Mitchell, J. E. (1999b). Cutting plane algorithms for integer programming. In *Encyclopedia of Optimization.* Kluwer Academic.

Mühlenbein, H. (1991). Evolution in time and space–the parallel genetic algorithm. In Rawlins, G. J. E., editor, *Foundations of Genetic Algorithms.* Morgan Kaufmann.

Mühlenbein, H. (1997). Genetic algorithms. In Aarts, E. and Lenstra, J. K., editors, *Local Search in Combinatorial Optimization.* John Wiley.

Mulvey, J. M., Vanderbei, R. J., and Zenios, S. A. (1995). Robust optimization of large-scale systems. *Operations Research*, 43(2).

Nemhauser, G. L. and Wolsey, L. A. (1989). Integer programming. In Nemhauser, G. L., Kan, A. H. G. R., and Todd, M. J., editors, *Optimization*, volume 1. North-Holland.

Nevins, J. L., Whitney, D. E., De Fazio, T. L., Edsall, A. C., Gustavson, R. E., Metzinger, R. W., and Dvorak, W. A. (1989). *Concurrent Design of Products and Processes: A Strategy for the Next Generation in Manufacturing.*

McGraw-Hill Publishing Company, New York.

Nwana, H. S. and Ndumu, D. T. (1997). An introduction to agent technology. In Nwana, H. S. and Azarmi, N., editors, *Software Agents and Soft Computing, Lecture Notes in Artificial Intelligence*, volume 1198. Springer.

Ortega, J., Bernier, J. L., Díaz, A. F., Rojas, I., Salmerón, M., and Prieto, A. (1999). Parallel combinatorial optimization with evolutionary cooperation between processors. In *Proceedings of the IEEE Congress on Evolutionary Computation*, Washington D. C.

Ortega, J. M. (1987). *Matrix Theory: A Second Course.* Plenum Press, New York.

Orvosh, D. and Davis, L. (1993). Shall we repair? genetic algorithms, combinatorial optimization and feasibility constraints. In Forrest, S., editor, *Proceedings of the Fifth International Conference on Genetic Algorithms.* Morgan Kaufmann.

Papadimitriou, C. H. and Steiglitz, K. (1982). *Combinatorial Optimization: Algorithms and Complexity.* Prentice-Hall, New Jersey.

Parker, R. G. and Rardin, R. L. (1988). *Discrete Optimization.* Academic Press, San Diego.

Pearl, J. (1984). *Heuristics: Intelligent Search Strategies for Computer Problem Solving.* Addison-Wesley, Massachusetts.

Peck, C. C. and Dhawan, A. P. (1995). Genetic algorithms as global random search methods: An alternative perspective. *Evolutionary Computation*, 3(1).

Pierreval, H. and Tautou, L. (1997). Using evolutionary algorithms and simulation for the optimization of manufacturing systems. *IIE Transactions*, 29(3).

Poli, C., Dastidar, P., and Graves, R. J. (1992). Design knowledge acquisition for DFM methodologies. *Research in Engineering Design*, 4.

Potter, M. A. (1997). *The Design and Analysis of a Computational Model of Cooperative Coevolution.* PhD thesis, George Mason University, Fairfax, Virginia.

Potter, M. A. and De Jong, K. A. (2000). Cooperative coevolution: An architecture for evolving coadapted subcomponents. *Evolutionary Computation*, 8(1).

Powell, D. J. (1990). *Inter-GEN: A Hybrid Approach to Engineering Design Optimization.* PhD thesis, Rensselaer Polytechnic Institute, Troy, NY.

Prügel-Bennett, A. and Shapiro, J. L. (1994). An analysis of genetic algorithms using statistical mechanics. *Physics Review Letters*, 72.

Qi, X. and Palmieri, F. (1994). Theoretical analysis of evolutionary algorithms with an infinite population size in continuous space, Part-I: Basic properties of selection and mutation. *IEEE Transactions on Neural Networks*, 5(1).

Raidl, G. R. (1998). An improved genetic algorithm for the multiconstrained 0-1 knapsack problem. In *Proceedings of the World Conference on Computational Intelligence*, Anchorage, Alaska.

Richardson, J. T., Palmer, M. R., Liepins, G., and Hilliard, M. (1989). Some guidelines for genetic algorithms with penalty functions. In Schaffer, J., editor, *Proceedings of the Third International Conference on Genetic Algo-*

rithms. Morgan Kaufmann.

Roberts, F. S. (1984). *Applied Combinatorics*. Prentice-Hall, New Jersey.

Rosin, C. D. (1997). *Coevolutionary Search Among Adversaries*. PhD thesis, University of California, San Diego, San Diego, California.

Rudolph, G. (1994). Convergence analysis of canonical genetic algorithms. *IEEE Transactions on Neural Networks*, 5(1).

Rudolph, G. (1996). Convergence of evolutionary algorithms in general search spaces. In *Proceedings of the IEEE International Conference on Evolutionary Computation*, Nagoya, Japan.

Russell, S. J. and Norvig, P. (1995). *Artificial Intelligence: A Modern Approach*. Prentice Hall, New Jersey.

Sanderson, A. C., Graves, R. J., and Millard, D. L. (1994). Multipath agility in electronics manufacturing. In *Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics*.

Schnecke, V. and Vornberger, O. (1997). Hybrid genetic algorithms for constrained placement problems. *IEEE Transactions on Evolutionary Computation*, 1(4).

Seredynski, F. (1997). Competitive coevolutionary multi-agent systems: The application to mapping and scheduling problems. *Journal of Parallel and Distributed Computing*, 47(1).

Shapiro, J., Prügel-Bennett, A., and Rattray, M. (1994). A statistical mechanical formulation of the dynamics of genetic algorithms. In *Lecture Notes in Computer Science*, volume 865. Springer, Berlin.

Sikdar, B., Kalyanaraman, S., and Vastola, K. S. (2000). An integrated model for the latency and steady state throughput of TCP connections. In *Proceedings of the IFIP Symposium on Advanced Performance Modeling*, Orlando, Florida.

Sikora, R. and Shaw, M. J. (1997). Coordination mechanisms for multi-agent manufacturing systems: Applications to integrated manufacturing scheduling. *IEEE Transactions on Engineering Management*, 44(2).

Slatkin, M. (1970). Selection and polygenic characters. *Proceedings of the National Academy of Sciences*, 66(1).

Smith, R. E. (1997). Heuristics for parameter-setting issues: Population size. In Bäck, T., Fogel, D., and Michalewicz, Z., editors, *Handbook of Evolutionary Computation*. Oxford University Press.

Smith, R. G. (1980). The contract net protocol: High-level communication and control in a distributed problem solver. *IEEE Transactions on Computers*, C-29(12).

Smith, R. G. and Davis, R. (1981). Frameworks for cooperation in distributed problem solving. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-11(1).

Sobolewski, M. W. and Erkes, J. W. (1995). Camnet: Architecture and applications. In *Proceedings of Concurrent Engineering 1995 Conference*, McLean, Virginia.

Song, L. and Nagi, R. (1997). Design and implementation of a virtual information system for agile manufacturing. *IIE Transactions*, 29(10).

Spears, W. M. (1997). Heuristics for parameter-setting issues: Recombination parameters. In Bäck, T., Fogel, D., and Michalewicz, Z., editors, *Handbook of Evolutionary Computation*. Oxford University Press.

Spears, W. M. and De Jong, K. A. (1996). Analyzing GAs using markov models with sematically ordered and lumped states. In *Proceedings of the Foundations of Genetic Algorithms*.

Srinivas, M. and Patnaik, L. M. (1996). Genetic search: Analysis using fitness moments. *IEEE Transactions on Knowledge and Data Engineering*, 8(1).

Stephens, C. and Waelbroeck, H. (1999). Schemata evolution and building blocks. *Evolutionary Computation*, 7(2).

Stroustrup, B. (1991). *The C++ Programming Language*. Addison-Wesley, Massachusetts.

Subbu, R. and Bonissone, P. P. (2003). A retrospective view of fuzzy control of evolutionary algorithm behavior. In *Proceedings of the 2003 IEEE International Conference on Fuzzy Systems*, St. Louis, Missouri.

Subbu, R., Hocaoğlu, C., and Sanderson, A. C. (1998a). A virtual design environment using evolutionary agents. In *Proceedings of the IEEE International Conference on Robotics and Automation*, Leuven, Belgium.

Subbu, R. and Sanderson, A. C. (2000). Modeling and convergence analysis of distributed coevolutionary algorithms. In *Proceedings of the IEEE Congress on Evolutionary Computation*, San Diego, California.

Subbu, R. and Sanderson, A. C. (2001a). Network-based distributed planning for design and manufacturing. In *Proceedings of the IEEE International Symposium on Assembly and Task Planning*, Fukuoka, Japan.

Subbu, R. and Sanderson, A. C. (2001b). Network distributed virtual design using coevolutionary agents. In *Proceedings of the IEEE International Conference on Robotics and Automation*, Seoul, Korea.

Subbu, R. and Sanderson, A. C. (2003a). Modeling and convergence analysis of distributed coevolutionary algorithms. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*. In Press.

Subbu, R. and Sanderson, A. C. (2003b). Network-based distributed planning using coevolutionary agents: Architecture and evaluation. *IEEE Transactions on Systems, Man, and Cybernetics, Part A: Systems and Humans*. In Press.

Subbu, R., Sanderson, A. C., Hocaoğlu, C., and Graves, R. J. (1998b). Distributed virtual design environment using intelligent agent architecture. In *Proceedings of the Industrial Engineering Research Conference*, Banff, Canada.

Subbu, R., Sanderson, A. C., Hocaoğlu, C., and Graves, R. J. (1998c). Evolutionary intelligent agents for distributed virtual design. In *Proceedings of Rensselaer's International Conference on Agile, Intelligent, and Computer Integrated Manufacturing*, Troy, New York.

Subbu, R., Sanderson, A. C., Hocaoğlu, C., and Graves, R. J. (1999). Evolutionary decision support for distributed virtual design in modular product manufacturing. *Production Planning and Control*, 10(7). Special Issue on Agile, Intelligent, and Computer Integrated Manufacturing.

Sycara, K., Roth, S., Sadeh, N., and Fox, M. (1991). Distributed constrained heuristic search. *IEEE Transactions on Systems, Man, and Cybernetics*, 21(6).

Talukdar, S. N. and de Souza, P. (1995). Insects, fish, and computer-based super-agents. In Chow, J. H., Kokotovic, P. V., and Thomas, R. J., editors, *Systems and Control Theory for Power Systems, The IMA Volumes in Mathematics and its Applications*, volume 64. Springer-Verlag.

Tanese, R. (1989). *Distributed Genetic Algorithms for Function Optimization.* PhD thesis, University of Michigan, Ann Arbor, Michigan.

Tappeta, R. V., Renaud, J. E., and Rodriguez, J. F. (2002). An interactive multiobjective optimization design strategy for decision based multidisciplinary design. *Engineering Optimization*, 34(5).

Taylor, D. and Graves, R. J. (1990). An examination of routeing flexibility for small batch assembly of printed circuit boards. *International Journal of Production Research*, 28(11).

Taylor, G. D. (1997). Design for global manufacturing and assembly. *IIE Transactions*, 29(7).

Tong, S. S., Powell, D., and Cornett, D. (1992). Engineous: A unified method for design automation, optimization, and integration. In Tong, C. and Sriram, D., editors, *Artificial Intelligence in Engineering Design*, volume 3. Academic Press.

Trojanowski, K. and Michalewicz, Z. (1999). Searching for optima in nonstationary environments. In *Proceedings of the IEEE Congress on Evolutionary Computation*, Washington D. C.

Vose, M. D. and Wright, A. H. (1995). Simple genetic algorithms with linear fitness. *Evolutionary Computation*, 2(4).

Weicker, K. and Weicker, N. (1999). On evolution strategy optimization in dynamic environments. In *Proceedings of the IEEE Congress on Evolutionary Computation*, Washington D. C.

Wiegand, R. P., Liles, W. C., and De Jong, K. A. (2002). Analyzing cooperative coevolution with evolutionary game theory. In *Proceedings of the World Conference on Computational Intelligence*, Honolulu, Hawaii.

Wilson, J. M. (1997). A genetic algorithm for the generalized assignment problem. *Journal of the Operational Research Society*, 48(8).

Wolpert, D. H. and MacReady, W. G. (1997). No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1).

Wright, A. H. (1999). The exact schema theorem. *Web published at: http://www.cs.umt.edu/u/wright/wright.htm.*

Yu, C.-Y. and Huang, H.-P. (2001). Development of the order fulfillment process in the foundry fab by applying distributed multi-agents on a generic message-passing platform. *IEEE/ASME Transactions on Mechatronics*, 6(4).

Zhigljavsky, A. A. (1991). *Theory of Global Random Search.* Kluwer Academic Publishers, Dordrecht, Netherlands.

Index

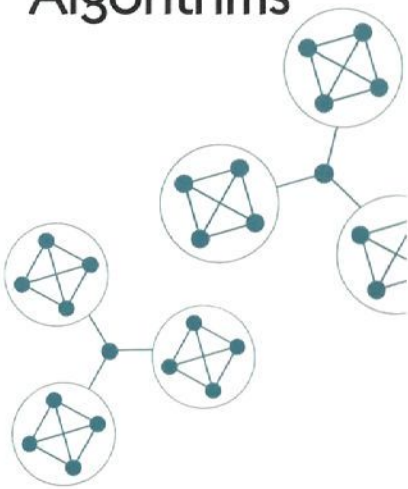
In this book, efficient and scalable coevolutionary algorithms for distributed, network-based decision-making, which utilize objective functions are developed in a networked environment where internode communications are a primary factor in system performance.

A theoretical foundation for this class of coevolutionary algorithms is introduced using techniques from stochastic process theory and mathematical analysis.

A case study in distributed, network-based decision-making presents an implementation and detailed evaluation of the coevolutionary decision-making framework that incorporates distributed evolutionary agents and mobile agents.

The methodology discussed in this book can have a fundamental impact on the principles and practice of engineering in the distributed, network-based environment that is emerging within and among corporate enterprise systems. In addition, the conceptual framework of the approach to distributed decision systems described may have much wider implications for network-based systems and applications.

Network-Based Distributed Planning Using Coevolutionary Algorithms



World Scientific
www.worldscientific.com
5470 hc

ISBN 981-238-754-4
9 789812 387547