Kluwer's
International Series

ADVANCING THE
STATE-OF-THE-ART

Volume Contributors:

James C. Bean
Yixin Chen
Carlos A. Coello Coello
Kalyanmoy Deb
Agoston E. Eiben
Elham Ghashghai
Tushar Goyal
Emma Hart
Frederick Hillier
Mark Hillier
Robert Hinterding
Jeffrey A. Joines
Michael G. Kay
An Li
Alex Z.-Z. Lin
Zbigniew Michalewicz
Bryan A. Norman
Ronald L. Rardin
Peter Ross
Thomas Runarsson
Ruhul Sarker
Martin Schmidt
Alice E. Smith
Benjamin Wah
Ingo Wegener
Chelsea C. White III
Kit Po Wong
Xin Yao

EVOLUTIONARY OPTIMIZATION

edited by
Ruhul Sarker
Masoud Mohammadian
Xin Yao

EVOLUTIONARY OPTIMIZATION

INTERNATIONAL SERIES IN
OPERATIONS RESEARCH & MANAGEMENT SCIENCE
Frederick S. Hillier, Series Editor Stanford University

Jaiswal, N.K. / *MILITARY OPERATIONS RESEARCH: Quantitative Decision Making*
Gal, T. & Greenberg, H. */ ADVANCES IN SENSITIVITY ANALYSIS AND PARAMETRIC PROGRAMMING*
Prabhu, N.U. / *FOUNDATIONS OF QUEUEING THEORY*
Fang, S.-C., Rajasekera, J.R. & Tsao, H.-S.J. / *ENTROPY OPTIMIZATION AND MATHEMATICAL PROGRAMMING*
Yu, G. */ OPERATIONS RESEARCH IN THE AIRLINE INDUSTRY*
Ho, T.-H. & Tang, C. S. / *PRODUCT VARIETY MANAGEMENT*
El-Taha, M. & Stidham , S. */ SAMPLE-PATH ANALYSIS OF QUEUEING SYSTEMS*
Miettinen, K. M. / *NONLINEAR MULTIOBJECTIVE OPTIMIZATION*
Chao, H. & Huntington, H. G. / *DESIGNING COMPETITIVE ELECTRICITY MARKETS*
Weglarz, J. / *PROJECT SCHEDULING: Recent Models, Algorithms & Applications*
Sahin, I. & Polatoglu, H. / *QUALITY, WARRANTY AND PREVENTIVE MAINTENANCE*
Tavares, L. V. */ ADVANCED MODELS FOR PROJECT MANAGEMENT*
Tayur, S., Ganeshan, R. & Magazine, M. / *QUANTITATIVE MODELING FOR SUPPLY CHAIN MANAGEMENT*
Weyant, J./ *ENERGY AND ENVIRONMENTAL POLICY MODELING*
Shanthikumar, J.G. & Sumita, U./*APPLIED PROBABILITY AND STOCHASTIC PROCESSES*
Liu, B. & Esogbue, A.O. / *DECISION CRITERIA AND OPTIMAL INVENTORY PROCESSES*
Gal, T., Stewart, T.J., Hanne, T./ *MULTICRITERIA DECISION MAKING: Advances in MCDM Models, Algorithms, Theory, and Applications*
Fox, B. L./ *STRATEGIES FOR QUASI-MONTE CARLO*
Hall, R.W. / *HANDBOOK OF TRANSPORTATION SCIENCE*
Grassman, W.K./ *COMPUTATIONAL PROBABILITY*
Pomerol, J-C. & Barba-Romero, S. / *MULTICRITERION DECISION IN MANAGEMENT*
Axsäter, S./ *INVENTORY CONTROL*
Wolkowicz, H., Saigal, R., Vandenberghe, *L./ HANDBOOK OF SEMI-DEFINITE PROGRAMMING: Theory, Algorithms, and Applications*
Hobbs, B. F. & Meier, P. / *ENERGY DECISIONS AND THE ENVIRONMENT: A Guide to the Use of Multicriteria Methods*
Dar-El, *E./ HUMAN LEARNING: From Learning Curves to Learning Organizations*
Armstrong, J. S./ *PRINCIPLES OF FORECASTING: A Handbook for Researchers and Practitioners*
Balsamo, S., Personé, V., Onvural, R./ *ANALYSIS OF QUEUEING NETWORKS WITH BLOCKING*
Bouyssou, D. et al/ *EVALUATION AND DECISION MODELS: A Critical Perspective*
Hanne, T./ *INTELLIGENT STRATEGIES FOR META MULTIPLE CRITERIA DECISION MAKING*
Saaty, T. & Vargas, L./ *MODELS, METHODS, CONCEPTS & APPLICATIONS OF THE ANALYTIC HIERARCHY PROCESS*
Chatterjee, K. & Samuelson, W./ *GAME THEORY AND BUSINESS APPLICATIONS*
Hobbs, B. et al/ *THE NEXT GENERATION OF ELECTRIC POWER UNIT COMMITMENT MODELS*
Vanderbei, R. J./ *LINEAR PROGRAMMING: Foundations and Extensions, 2nd Ed.*
Kimms, A./ *MATHEMATICAL PROGRAMMING AND FINANCIAL OBJECTIVES FOR SCHEDULING PROJECTS*
Baptiste, P., Le Pape, C. & Nuijten, W./ *CONSTRAINT-BASED SCHEDULING*
Feinberg, E. & Shwartz, A./ *HANDBOOK OF MARKOV DECISION PROCESSES: Methods and Applications*
Ramík, J. & Vlach, M. / *GENERALIZED CONCAVITY IN FUZZY OPTIMIZATION AND DECISION ANALYSIS*
Song, J. & Yao, D. / *SUPPLY CHAIN STRUCTURES: Coordination, Information and Optimization*
Kozan, E. & Ohuchi, A./ *OPERATIONS RESEARCH/ MANAGEMENT SCIENCE AT WORK*
Bouyssou et al/ *AIDING DECISIONS WITH MULTIPLE CRITERIA: Essays in Honor of Bernard Roy*
Cox, Louis Anthony, Jr./ *RISK ANALYSIS: Foundations, Models and Methods*
Dror, M., L'Ecuyer, P. & Szidarovszky, F. / *MODELING UNCERTAINTY: An Examination of Stochastic Theory, Methods, and Applications*
Dokuchaev, N./ *DYNAMIC PORTFOLIO STRATEGIES: Quantitative Methods and Empirical Rules for Incomplete Information*

EVOLUTIONARY OPTIMIZATION

Edited by
RUHUL SARKER
University of New South Wales

MASOUD MOHAMMADIAN
University of Canberra

XIN YAO
University of Birmingham

KLUWER ACADEMIC PUBLISHERS
NEW YORK, BOSTON, DORDRECHT, LONDON, MOSCOW

eBook ISBN: 0-306-48041-7
Print ISBN: 0-7923-7654-4

Created in the United States of America

Visit Kluwer Online at: http://kluweronline.com
and Kluwer's eBookstore at: http://ebooks.kluweronline.com

Contents

Preface

Evolutionary computation techniques have attracted increasing attentions in recent years for solving complex optimization problems. They are more robust than traditional methods based on formal logics or mathematical programming for many real world OR/MS problems. Evolutionary computation techniques can deal with complex optimization problems better than traditional optimization techniques. However, most papers on the application of evolutionary computation techniques to Operations Research /Management Science (OR/MS) problems have scattered around in different journals and conference proceedings. They also tend to focus on a very special and narrow topic. It is the right time that an archival book series publishes a special volume which includes critical reviews of the state-of-art of those evolutionary computation techniques which have been found particularly useful for OR/MS problems, and a collection of papers which represent the latest development in tackling various OR/MS problems by evolutionary computation techniques. This special volume of the book series on Evolutionary Optimization aims at filling in this gap in the current literature.

The special volume consists of invited papers written by leading researchers in the field. All papers were peer reviewed by at least two recognised reviewers. The book covers the foundation as well as the practical side of evolutionary optimization.

This book contains 17 chapters which can be categorized into the following seven parts:

1 Introduction

2 Single Objective Optimization

3 Multiobjective Optimization

4 Hybrid Algorithms

5 Parameter Selection

6 Application of EAs to Practical Problems

7 Application of EAs to Theoretical Problems

This book will be useful to postgraduate course work students, researchers, doctoral students, instructors and practitioners in OR/MS, computer science, industrial engineering, business, and applied mathematics. We expect that the promising opportunities illustrated by the case studies and the tools and techniques described in the book will help to expand the horizons of evolutionary optimization and disseminate knowledge to both the research and the practice communities.

We would like to thank Prof. Fred Hillier of Standford University (series editor for International Series in Operations Research and Management Science for Kluwer Academic Publishers) and Prof. Fred Glover of University of Colorado, USA, for their advice in preparing the proposal of the book. We are grateful to the unknown reviewers for the book proposal for their constructive and useful suggestions.

We would like to acknowledge the help of all involved in the collation and the review process of the book, without whose support the project could not have been satisfactorily completed. Most of the authors of chapters included in this volume also served as referees for articles written by other authors. Thanks also to several other referees who have kindly refereed chapters accepted for in this book. Thanks go to all those who provided constructive and comprehensive reviews and comments. A further special note of thanks goes to all the staff at Kluwer Academic Publisher, whose contribution throughout the whole process from inception to final publication have been invaluable. Ruhul Sarker also likes to thank Dr W. Zhu and Dr H. Abbass for their help in initial formatting using LaTex.

In closing, we wish to thank all the authors for their insight and excellent contributions to this book. In addition, this book would not have been possible without the ongoing professional support from Mr. Gary Folven, Publisher in OR/MS and Ms. Deborah Doherty, Electronic Production Manager, Kluwer Academic Publishers. Finally, we want to thank our families for their love and support throughout this project.

R. SARKER, M. MOHAMMADIAN AND X. YAO

Contributing Authors

James C. Bean, Professor and Associate Dean for Graduate Education, University of Michigan, College of Engineering, 1221 Beal Ave, Ann Arbor, MI 48109-2102, Email: jbean@engin.umich.edu

Yixin Chen, Department of Computer Science and the Coordinated Science Laboratory, Univ. of Illinois at Urbana-Champaign, 447 CSRL, 1304 West Main Street, Urbana, IL 61801.
Email: chen@manip.crhc.uiuc.edu

Carlos A. Coello Coello, CINVESTAV-IPN, Departamento de Ingenieria Electrica Seccion de Computacion, Av. Instituto Politecnico Nacional No. 2508, Col. San Pedro Zacatenco, Mexico, D.F. 07300, Email: ccoello@cs.cinvestav.mx

Kalyanmoy Deb, Professor, Kanpur Genetic Algorithms Laboratory, Department of Mechanical Engineering, Indian Institute of Technology Kanpur, Kanpur, Pin 208 016, INDIA, Email: deb@iitk.ac.in

Agoston E. Eiben, Faculty of Sciences, Vrije Universiteit Amsterdam, De Boelelaan 1081a, 1081 HV Amsterdam, The Netherlands.

Elham Ghashghai, Engineer, Rand Corporation, 1700 Main Street, PO Box 2138, Santa Monica, CA 90407-2138, Email: elham@rand.org

Tushar Goyal, Department of Mech Engg, Indian Institute of Technology, Kanpur, Pin 208 016, INDIA, Email: tusharg_99@yahoo.com

Emma Hart, School of Computing, Napier University, 219 Colinton Road, Edinburgh, EH14 1DJ UK.

Frederick Hillier, Management Science and Engineering, Terman Engineering Center, 3rd Floor, Stanford University, Stanford, California 94305-4026, Email: fhillier@leland.Stanford.EDU

Mark Hillier, Associate Professor, Dept. of Management Science, University of Washington, Mackenzie 370, Box 353200 Seattle, WA 98195-3200, Email: mhillier@mac.com

Robert Hinterding, Department of Computer and Mathematical Sciences, Victoria University of Technology, PO Box 14428 MCMC Melbourne 8001, Australia.

Jeffrey A. Joines, Assistant Professor, Textile Engineering, Chemistry, and Science, North Carolina State University, Campus Box 8301, Raleigh, N.C. 27695-8301, Email: jjoine@eos.ncsu.edu

Michael G. Kay, Industrial Engineering, North Carolina State University, Campus Box 7906, Raleigh, N.C. 27695-7906.

An Li, Artificial Intelligence and Power Systems Research Group (AIPS), Department of EEE, The University of Western Australia, 35 Stirling Highway, CRAWLEY, WA 6009, Australia, Email: leon@ee.uwa.edu.au

Alex Z.-Z. Lin, Industrial and Operations Engineering, Univ. of Michigan at Ann Arbor, 1221 Beal Ave, Ann Arbor, MI 48109-2102.

Zbigniew Michalewicz, NuTech Solutions, Inc., 8401 University Executive Park, Suite 102, Charlotte, NC 28262 USA.

Bryan A. Norman, Assistant Professor, Industrial Engineering Department, University of Pittsburgh, 1033 Benedum Hall, Pittsburgh, PA 15261, Email: banorman@engrng.pitt.edu

Ronald L Rardin, Professor, Industrial Engineering, Purdue University, West Lafayette, IN 47907-1287, Email: rardin@ecn.purdue.edu

Peter Ross, Professor, School of Computing, Napier University, 219 Colinton Road, Edinburgh EH14 1DJ UK, Email: peter@dcs.napier.ac.uk

Thomas Runarsson, Department of Mechanical Engineering, University of Iceland, Hjardarhagi 2-6, IS-107 Reykjavik, Iceland, Email: tpr@verk.hi.is

Ruhul Sarker, School of Computer Science, University of New South Wales, ADFA Campus, Northcott Drive, Canberra 2600, Australia. Email: ruhul@cs.adfa.edu.au

Martin Schmidt, NuTech Solutions, Inc., 8401 University Executive Park, Suite 102, Charlotte, NC 28262 USA.

Alice E. Smith, Professor and Chair, Department of Industrial and Systems Engineering, 207 Dunstan Hall, Auburn University, AL 36849-5346 USA, Email: aesmith@eng.auburn.edu

Benjamin Wah, Professor, Department of Computer Science, Univ. of Illinois at Urbana-Champaign, 447 CSRL, 1304 West Main Street, Urbana, IL 61801, Email: b-wah@uiuc.edu

Ingo Wegener, Professor, Universitaet Dortmund, Lehrstuhl Informatik 2, 44221 Dortmund, Germany.
Email: wegener@ls2.cs.uni-dortmund.de

Chelsea C. White III, Professor, Department of Industrial and Operations Engineering, 1205 Beal Avenue, The University of Michigan, Ann Arbor, MI 48109-2117.

Kit Po Wong, Professor, Artificial Intelligence and Power Systems Research Group (AIPS), Department of EEE, The University of Western Australia, 35 Stirling Highway, CRAWLEY, WA 6009, Australia, Email: kitpo@ee.uwa.edu.au

Xin Yao, Professor, School of Computer Science, The University of Birmingham, Edgbaston, Birmingham B15 2TT, U.K.
Email: X.Yao@cs.bham.ac.uk

List of Reviewers

H. Abbass, Australia
J. Bean, USA
C. Coello Coello, Mexico
N. Clapham, Australia
K. Deb, India
J. Joines, USA
L. Khan, Australia
M. Kirley, Australia
J. Knowles, UK
K. Liang, Australia
R. Mckay, Australia
Z. Michalewicz, USA
M. Mohammadian, Australia
C. Newton, Australia
R. Rardin, USA
P. Ross, UK
T. Runarsson, Iceland
R. Sarker, Australia
M. Schoenauer, France
A. Smith, USA
B. Wah, USA
I. Wegener, Germany
E. Zitzler, Switzerland

I

INTRODUCTION

This page intentionally left blank

Chapter 1

CONVENTIONAL OPTIMIZATION TECHNIQUES

Mark S. Hillier and
Frederick S. Hillier

Operations research (OR) and management science (MS) are disciplines that attempt to aid managerial decision making by developing mathematical models that describe the essence of a problem and then applying mathematical procedures to solve the models. The purpose of this chapter is to present an overview of conventional OR/MS techniques in optimization. This will include discussion of the various types of models that are used and the approaches that are used to solve them.

We first explore the nature of optimization models in general. The mathematical model of a business problem is a system of equations and mathematical expressions that describe the essence of the problem. If there are n quantifiable decisions to be made, they are represented by *decision variables* (say, $x_1, x_2, \ldots, x_n$). An appropriate performance measure (e.g., profit) is then defined as a function of the decision variables (e.g., Profit $= 2x_1 + 5x_2$). This function is called the *objective function.* If there are restrictions on the values that the decision variables can take, these are expressed mathematically. These restrictions are typically expressed as inequalities (e.g., $x_1 + 2x_2 \leq 3$) or equations (e.g., $x_1 x_2 = 7$), and are called *constraints.* The goal is then to choose the values of the decision variables that achieve the best value of the objective function subject to satisfying each of the constraints.

For example, consider the problem of choosing the production level of n different products so as to maximize profit, subject to m restrictions on the production levels (e.g., due to limited resources). The model then is to choose $x_1, x_2, \ldots, x_n$ so as to

$$
\begin{aligned}
&\text{Maximize} && f(x_1, x_2, \ldots, x_n) \\
&\text{subject to} && \\
& && g_1(x_1, x_2, \ldots, x_n) \leq b_1 \\
& && g_2(x_1, x_2, \ldots, x_n) \leq b_2 \\
& && \quad \vdots \\
& && g_m(x_1, x_2, \ldots, x_n) \leq b_m \\
&\text{and} && \\
& && x_1 \geq 0, x_2 \geq 0, \ldots, x_n \geq 0.
\end{aligned} \tag{1.1}
$$

In this model, the decision variables $(x_1, x_2, \ldots, x_n)$ represent the production levels, the objective function $f(x_1, x_2, \ldots, x_n)$ measures the profit, and the inxconstraints are represented by the inequalities $g_i(x_1, x_2, \ldots, x_n) \leq b_i$ for $i = 1, \ldots, m$.

Any choice of values of $(x_1, x_2, \ldots, x_n)$ is called a *solution,* whereas a solution satisfying all the constraints is a *feasible solution.* The set of all feasible solutions is called the *feasible region.* A solution in the feasible region that maximizes the objective function is called an *optimal solution.*

Other optimization models may have the goal of *minimizing* the objective function (e.g., minimizing cost). Also, models may have a mixture of constraints, some with $\leq$ signs, some with $\geq$ signs, and some with = signs, or some may have no constraints at all.

1. Classifying Optimization Models

Before solving an optimization model, it is important to consider the form and mathematical properties of the objective function, constraints, and decision variables. For example, the objective function might be linear or nonlinear, differentiable or nondifferentiable, concave or convex, etc. The decision variables might be continuous or discrete. The feasible region might be convex or nonconvex. These differences each impact how the model can be solved, and thus optimization models are classified according to these differences. This section defines a number of mathematical properties, and then classifies optimization models according to these properties.

Many optimization solution techniques depend upon the objective function and/or the functions in the constraints being *linear functions.* A function is linear if it can be expressed in the form $f(x_1, x_2, \ldots, x_n) = c_1x_1 + c_2x_2 + \cdots + c_nx_n$, where the c_i are constants. If the objective

function and all the constraint functions are linear functions, then the model is called a *linear programming model.*

In (1.1), there was a single goal—to maximize profit. Sometimes, however, it is not possible to include all the managerial objectives within a single overriding objective, such as maximizing profit or minimizing cost. For these reasons, models may include *multiple* objective functions and analysis of the problem may require individual consideration of the separate objectives. *Goal programming* provides a way of striving toward several objectives simultaneously.

Linear programming and nonlinear programming techniques conventionally assume that the decision variables are *continuous.* That is, the decision variables are allowed to have *any* value that satisfies the constraints, including noninteger values. In many applications, however, the decision variables make sense only if they have integer values. For example, it is often necessary to assign people or equipment in integer quantities. Thus, some optimization models include *discrete* decision variables, usually restricting some or all of the decision variables to integer values. A model where all the variables are required to be integer is called an *integer programming model.* If some of the variables are continuous, but others are required to be integer, it is called a *mixed-integer programming model.* An integer programming model can be further categorized as either an integer linear programming model (if all the functions in the model are linear) or an integer nonlinear programming model (if any of the functions are nonlinear). It is common when identifying an integer linear programming model to drop the adjective linear.

A model where either the objective function or any of the constraints includes a nonlinear function is called a *nonlinear programming model.* Nonlinear programming problems come in many different shapes and forms. No single algorithm can solve all these different types of problems. Instead, algorithms have been developed to solve various individual classes (special types) of nonlinear programming problems. For example, the objective function can be concave, convex, or neither, differentiable or nondifferentiable, quadratic or not, and so on. The constraints can be linear or nonlinear, or the problem can be unconstrained. The feasible region can be a convex set or a nonconvex set. Many of these terms and classifications are defined in Section 5, and various algorithms that are used to solve these different classes of nonlinear programming models will be discussed.

In the following sections, we discuss linear programming, goal programming, integer programming, nonlinear programming, and simulation. This encompasses most of the important types of optimization models (or at least those types where evolutionary optimization algo-

rithms might be applicable). We conclude with suggestions for further reading if more detailed information about conventional optimization techniques is desired.

2. Linear Programming

Linear programming is one of the most widely used techniques of operations research and management science. Its name means that planning ("*programming*") is being done with a mathematical model (called a *linear programming model*) where all the functions in the model are *linear* functions.

Consider the example that resulted in (1.1). If the contribution of each product to profit is proportional to the level of production x_j, then each product contributes $c_j x_j$ to profit (where c_j is a constant). Further suppose that b_i is the available quantity of resource i, and that the usage of each resource i is proportional to the level of production x_j and equal to $a_{ij}x_j$ (where a_{ij} is a constant). The resulting linear programming model is then to choose $x_1, x_2, \ldots, x_n$ so as to

$$\begin{aligned} \text{Maximize} \quad & c_1x_1 + c_2x_2 + \cdots + c_nx_n \\ \text{subject to} \quad & \\ & a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n \le b_1 \\ & a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n \le b_2 \\ & \qquad\qquad \vdots \\ & a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n \le b_m \\ \text{and} \quad & \\ & x_1 \ge 0, x_2 \ge 0, \ldots, x_n \ge 0. \end{aligned} \tag{1.2}$$

Another common form for a linear programming model is to *minimize* the objective function, subject to functional constraints with $\ge$ signs and nonnegativity constraints. A typical interpretation then is that the objective function represents the total cost for the chosen mix of activities and the functional constraints involve different kinds of benefits. In particular, the function on the left-hand side of each functional constraint gives the level of a particular kind of benefit that is obtained from the mix of activities, and the constant on the right-hand side represents the minimum acceptable level of the benefit. Still other linear programming models have an equality instead of inequality sign in some or all of the functional constraints. Such constraints represent fixed requirements for the value of the function on the left-hand side. It is also fairly common for large linear programming models to include a mixture of functional

constraints—some with $\leq$ signs, some with $\geq$ signs, and some with = signs.

2.1 Some Applications of Linear Programming

Linear programming has been applied to a wide variety of problems. These applications include determining the best mix of products to produce given a limited set of resources, personnel scheduling to minimize cost while meeting service requirements, mixing a blend of raw materials to meet various requirements, and many, many more.

A number of important applications fit a special type of linear programming model called the *minimum-cost flow problem.* A typical application of this type involves shipping goods through a distribution network from certain supply points to certain demand points. Given the supply available from each supply point, the amount needed at each demand point, and the maximum amount that can be shipped through each branch of the distribution network, the objective is to determine how to route the goods so as to minimize the total shipping cost.

Some other important applications of linear programming are special cases of the minimum-cost flow problem. One special case, called the *transportation problem,* involves direct shipments from the supply points to the demand points (where the only limits on shipment amounts are the supplies and demands at these points), so the only decisions to be made are how much to ship from each supply point to each demand point. A second special case, called the *assignment problem,* involves assigning people (or machines or vehicles or plants) to tasks so as to minimize the total cost or time for performing these tasks. The *shortest path problem,* which involves finding the shortest route (in distance, cost, or time) through a network from an origin to a destination, also is a special case. Still another special case, called the *maximum flow problem,* is concerned with how to route goods through a distribution network from a single supply point to a single demand point so as to maximize the flow of goods without exceeding the maximum amount that can be shipped through each branch of the network.

2.2 Solving Linear Programming Models

In 1947, George Dantzig developed a remarkably efficient algorithm, called the *simplex method,* for finding an optimal solution for a linear programming model. The simplex method exploits some basic properties of optimal solutions for linear programming models. Because all the functions in the model are linear functions, the set of feasible solutions (the feasible region) is a convex set, as defined in Section 5. The vertices

of the feasible region play a special role in finding an optimal solution. A model will have an optimal solution if it has any feasible solutions and the constraints prevent improving the value of the objective function indefinitely. Any such model must have either exactly one optimal solution or an infinite number of them. In the former case, the one optimal solution must be a vertex of the feasible region. In the latter case, at least two vertices must be optimal solutions, and all the convex linear combinations of these vertices also are optimal. Therefore, it is sufficient to find the vertices with the most favorable value of the objective function in order to identify all optimal solutions.

Based on these facts, the simplex method is an iterative algorithm that only examines vertices of the feasible region. At each iteration, it uses algebraic procedures to move along an outside edge of the feasible region from the current vertex to an "adjacent" vertex that is better. The algorithm terminates when a vertex is reached that has no better adjacent vertices, because the convexity of the feasible region then implies that this vertex is optimal.

The simplex method is an *exponential-time algorithm.* That is, the solution time can theoretically grow exponentially in the number of variables and constraints. However, it consistently has proven to be very efficient in practice. Running time tends to grow approximately with the cube of the number of functional constraints, and less than linearly in the number of variables. Problems with many thousands of functional constraints and decision variables are routinely solved. Furthermore, continuing improvements in the computer implementation of the simplex method and its variants (particularly the *dual simplex method*) now are sometimes making it possible to solve massive problems ranging into the hundreds of thousands of functional constraints and millions of decision variables. One key to its efficiency on such large problems is that the path followed generally passes through only a tiny fraction of all vertices before reaching an optimal solution. The number of iterations (vertices traversed) generally is the same order of magnitude as the number of functional constraints.

Highly streamlined versions of the simplex method also are available to solve certain special types of linear programming problems in only a tiny fraction of the time that would be required by the general simplex method. For example, one such streamlined version is the *network simplex method,* which is widely used to solve minimum-cost flow problems. In addition, even more specialized algorithms are available to solve the special cases of the minimum-cost flow problem mentioned earlier,

namely, the transportation problem, the assignment problem, the shortest path problem, and the maximum flow problem.

In recent years, there has been a flurry of research to develop *interior-point methods.* The application of these methods to linear programming now has reached a high level of sophistication. These algorithms move through the interior of the feasible region until they converge to an optimal solution. A key feature of this approach is that both the number of iterations (trial solutions) and total running time tend to grow very slowly (even more slowly than for the simplex method) as the problem size is increased. Therefore, the best implementations tend to become faster than the simplex method for relatively large problems. However, this is not always true, because the efficiency of each approach depends greatly in different ways on the special structure in each individual problem.

3. Goal Programming

The models described in the other sections of this chapter assume that the objectives of the organization conducting the study can be encompassed within a single overriding objective, such as maximizing total profit or minimizing total cost, so that this overriding objective can be expressed in a single objective function for the model. However, this assumption is not always realistic. The management of some organizations frequently focus simultaneously on a wide variety of rather different objectives. In this case, a *multicriteria decision making* approach is needed.

A considerable number of multicriteria decision making techniques have been developed. We will briefly describe only one of these here, namely, a popular technique called *goal programming.*

The goal programming approach is to establish a specific numeric goal for each of the objectives, formulate an objective function for each objective, and then seek a solution that minimizes the total penalty assessed for missing these goals. This total penalty is expressed as a weighted sum of deviations of these objective functions from their respective goals. There are three possible types of goals. One is a *lower, one-sided goal* that sets a lower limit that we do not want to fall under (but exceeding the limit is fine), so a penalty is assessed only if the corresponding objective function falls below the goal. A second type is an *upper, one-sided goal* that sets an upper limit that we do not want to exceed (but falling under the limit is fine), so a penalty is assessed only if the corresponding objective function exceeds the goal. The third type is a *two-sided goal* that sets a specific target that we do not want to miss on either side,

so a penalty is assessed if the corresponding objective function deviates from the goal in either direction.

Goal programming problems can be categorized according to the conventional type of optimization model that it fits except for having multiple goals instead of a single objective. The most important case is *linear goal programming,* where all the objective functions and constraint functions are linear functions. In this case, it is possible to reformulate the linear goal programming model into a conventional linear programming model, so that the extremely efficient simplex method can be used to solve the model. One key to this reformulation is that the total penalty to be minimized can be expressed as a linear function of new variables that represent the relevant deviations from the respective goals.

Another categorization is according to how the goals compare in importance. In one case, called *nonpreemptive goal programming,* all the goals are of roughly comparable importance. In this case, when evaluating the total penalty for missing goals, the weights on the respective deviations are of the same order of magnitude. In another case, called *preemptive goal programming,* there is a hierarchy of priority levels for the goals, so that the goals of primary importance receive first-priority attention, those of secondary importance receive second-priority attention, and so forth (if there are more than two priority levels).

4. Integer Programming

One of the key assumptions of linear programming is that all the decision variables are *continuous variables,* so that either integer or noninteger values are allowed for these variables. However, many problems arise in practice where some or all of the decision variables need to be restricted to integer values. Integer programming is designed to deal with such problems.

The form of an integer (linear) programming model is identical to that shown in Section 2 for a linear programming model except that it has additional constraints specifying that certain decision variables must have an integer value. If every decision variable has such a constraint, the model is said to be a *pure* integer programming model, whereas it is a *mixed* integer programming model if only some of the decision variables have this constraint.

4.1 The Role of Binary Integer Programming Models

There have been numerous applications of integer programming that involve a direct extension of linear programming where the assumption of

continuous decision variables must be dropped. However, another area of application may be of even greater importance, namely, problems involving "yes-or-no decisions." In such decisions, the only two possible choices are *yes* and *no.* For example, should we undertake a particular fixed project? Should we make a particular fixed investment? Should we locate a facility on a particular site?

With just two choices, we can represent such decisions by decision variables that are restricted to just two values, say 0 and 1. Thus, the jth yes-or-no decision would be represented by, say, x_j such that

$$x_j = \begin{cases} 1, & \text{if decision } j \text{ is yes} \\ 0, & \text{if decision } j \text{ is no.} \end{cases}$$

Such variables are called *binary variables.* Consequently, integer programming models where all the integer-restricted variables are further restricted to be binary variables commonly are referred to as *binary integer programming* models (or *BIP* models for short). Such a model is a *pure* BIP model if all the variables are binary variables, whereas it is a *mixed* BIP model if only some of the variables are binary variables and the rest are continuous variables.

BIP models (either pure or mixed) are among the most widely used optimization models. We list below some examples of important types of BIP models, where the yes-or-no decisions represented by binary variables for each example are identified in parentheses.

- Capital budgeting with fixed investment proposals. (For each proposed investment, should it be made?)
- Site selection. (For each possible location of the sites for new facilities, should it be selected?)
- Designing a distribution network. (For each combination of a distribution center and a market area, should that distribution center be assigned to serve that market area?)
- Scheduling interrelated activities. (For each combination of an activity and a time period, should that activity begin in that time period?)
- Scheduling asset divestitures. (For each combination of an asset and a time period, should that asset be sold in that time period?)
- The fleet assignment problem. (For each combination of a type of airplane and a flight leg in the airline schedule, should that airplane type be assigned to that flight leg?)

- The crew scheduling problem. (For each combination of an airline crew and a flight leg in the airline schedule, should that crew be assigned to that flight leg?)

Such applications of BIP models often have provided very substantial savings for the companies involved. For example, annual savings of hundreds of millions of dollars have been achieved in the airline industry by applying BIP models to fleet assignment and crew scheduling problems.

4.2 Solving Integer Programming Models

The traditional approach that has been used to solve integer programming models with either general integer variables or binary variables is to apply an algorithm that is based on the *branch-and-bound technique.* The basic concept underlying this technique is to *divide and conquer.* Since the original "large" problem is too difficult to be solved directly, it is divided into smaller and smaller subproblems until these subproblems can be conquered. The dividing (*branching*) is done by partitioning the entire set of feasible solutions into smaller and smaller subsets. The conquering (*fathoming*) is done partially by *bounding* how good the best solution in the subset can be and then discarding the subset if its bound indicates that it cannot possibly contain an optimal solution for the original problem. This bounding commonly is done by using the simplex method or dual simplex method to solve the current subproblem's *LP-relaxation* (the linear programming model obtained by deleting the integer constraints from the subproblem).

The computational efficiency of branch-and-bound algorithms for integer programming is quite limited, and so is not at all comparable to the efficiency of the simplex method for linear programming. Such algorithms frequently fail to solve integer programming models with more than a hundred integer or binary variables.

Because many integer programming models arising in practice are too large to be solved by a branch-and-bound algorithm, research in recent years has focused on developing more efficient *branch-and-cut algorithms.* This kind of algorithm combines clever branch-and-bound techniques with two other kinds of techniques: *automatic problem preprocessing* and the *generation of cutting planes.* Automatic problem preprocessing involves a "computer inspection" of the user-supplied formulation of the integer programming model in order to spot reformulations that make the model quicker to solve without eliminating any feasible solutions. This involves identifying variables that can be fixed at one of their possible values, identifying and eliminating redundant constraints, and tightening some constraints without eliminating any feasible solu-

tions. Generating cutting planes involves introducing new functional constraints that reduce the feasible region for the LP-relaxation without eliminating any feasible solutions for the integer programming model. This is very helpful in providing tighter bounds when solving the LP-relaxation of either the original problem or any of its subproblems.

The new branch-and-cut algorithms have provided a rather dramatic improvement in computational efficiency for solving integer programming models. For example, they now are sometimes succeeding in solving BIP problems with many thousands of binary variables.

A significant amount of research is being conducted to develop algorithms for integer *nonlinear* programming. However, this is a very difficult problem and progress to date has been quite limited. No such algorithms have yet been adopted for widespread use in practice. There appears to be considerable potential for the application of evolutionary optimization algorithms in this area.

5. Nonlinear Programming

For most of the models in the preceding sections, it is assumed that all its functions (objective function and constraint functions) are linear. However, there are many practical problems for which this assumption does not hold. For instance, when there are economies of scale, the production function is nonlinear. In transportation problems, if there are volume discounts on shipping costs, then the shipping costs are nonlinear with respect to shipping volume. In portfolio selection problems, correlation between the performance of various securities causes the risk function to be nonlinear. For many problems the nonlinearities are small enough that it is reasonable to approximate them with linear functions. However, when the nonlinearities are not small, we often must deal directly with nonlinear programming models.

A general form for a nonlinear programming problem is to find $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ so as to

$$\begin{aligned} &\text{Maximize} && f(\mathbf{x}) \\ &\text{subject to} && \\ & && g_i(\mathbf{x}) \leq b_i \quad \text{for } i = 1, 2, \ldots, m \qquad (1.3) \\ &\text{and} && \\ & && \mathbf{x} \geq \mathbf{0}. \end{aligned}$$

Other forms are possible (e.g., minimization or $\geq$ constraints), but they can all be converted to (1.3), and for simplicity, we will assume this form in this section.

There are several complications that arise in nonlinear programming that do not arise in linear programming. First, unlike linear programming models, the optimal solution is not necessarily on the boundary of the feasible region. Therefore, a general algorithm for solving nonlinear programming models needs to consider *all* solutions in the feasible region, not just those on the boundary.

A second complication of nonlinear programming is that a local maximum (or minimum) need not be a global maximum (or minimum). Consider, for example, the single-variable function $f(x)$ plotted in Figure 1.1. This function has three local maxima (at $x = 1, x = 2$, and $x = 4$), but only one ($x = 4$) is a global maximum.

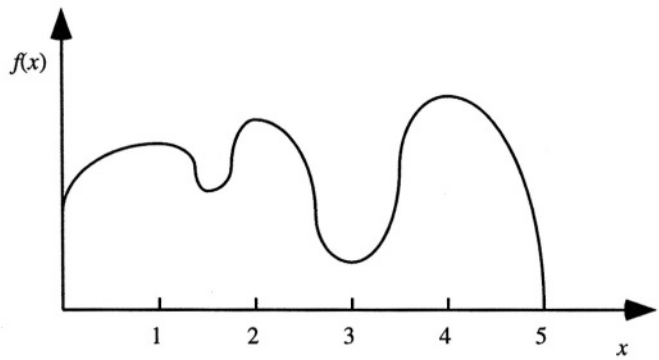


Figure 1.1. A function with several local maxima.

Most nonlinear programming algorithms search locally for improved solutions and hence can get "trapped" at a local maximum, with no way to guarantee that it is the global maximum. However, under some conditions, a local maximum is guaranteed to be a global maximum.

In a model that has no constraints, the objective function being *concave* guarantees that a local maximum is a global maximum. Similarly, the objective function being *convex* guarantees that a local minimum is a global minimum. A concave function is a function that is always "curving downward" (or not curving at all). A convex function is one that is always "curving upward" (or not curving at all). More specifically, $f(\mathbf{x})$ is a *convex function* if for each pair of points on the graph of the function, the line segment joining the two points lies on or above the graph. It is a *concave function* if for each pair of points on the graph of the function, the line segment joining the two points lies on or below the graph. Examples of both a concave function and a convex function are shown in Figure 1.2. A linear function is both concave and convex. The function shown in Figure 1.1 is neither concave nor convex.

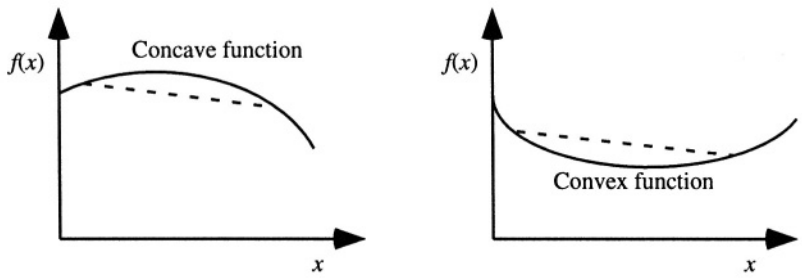


Figure 1.2. A concave function and a convex function.

However, if there are constraints, then even a model with a concave objective function can have a local maximum that is not a global maximum (or a convex objective function can have a local minimum that is not a global minimum). Consider a two-variable problem where the objective function is to maximize $5x_1 + 3x_2$, with the feasible region shown in Figure 1.3. This problem has local maxima at both (3, 4) and (7, 0), but only (7, 0) is a global maximum.

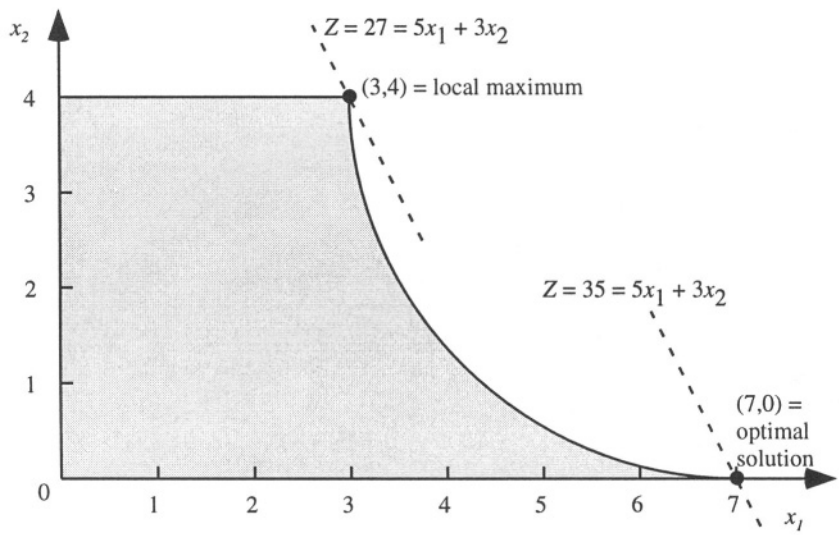


Figure 1.3. A nonconvex feasible region with multiple local optima.

For a constrained optimization problem, more is needed to guarantee that a local maximum is a global maximum. In particular, the feasible region must be a *convex set.* A convex set is a set of points such that the line segment connecting any pair of points in the set lies entirely within the set. Examples of both a convex set and a nonconvex set are shown

in Figure 1.4. If $f(\mathbf{x})$ is concave and the feasible region forms a convex set, then any local maxima must also be a global maximum.

If all the functions $g_i(\mathbf{x})$ [for the constraints $g_i(\mathbf{x}) \leq b_i$] are convex functions, then the resulting feasible region will be a convex set. Thus, a linearly constrained model will have a convex feasible region. However, the feasible region in Figure 1.3 is clearly nonconvex.

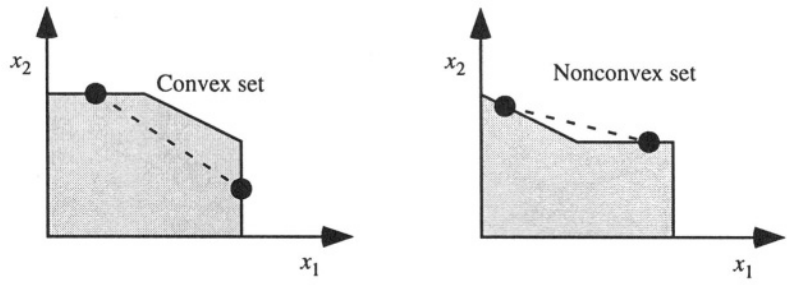


Figure 1.4. A convex set and a nonconvex set.

5.1 Types of Nonlinear Programming Models

Nonlinear programming problems come in many different shapes and forms. No single algorithm can solve all these different types of problems. Therefore, nonlinear programming problems are classified according to the properties of the objective function and constraints (if any).

An *unconstrained nonlinear programming model* has no constraints. Thus, the goal is simply to maximize $f(\mathbf{x})$ over all values of the decision variables $\mathbf{x} = (x_1, x_2, \ldots, x_n)$.

Linearly constrained nonlinear programming models have linear constraints, but the objective function is nonlinear. A number of special algorithms have been developed for this case, including a few that extend the simplex method to consider the nonlinear objective function.

An important special type of linearly constrained nonlinear model is a *quadratic programming model.* Quadratic programming models again have linear constraints, but are characterized by an objective function that is quadratic. That is, each term in the objective function consists of a constant times either a single decision variable, the square of a single decision variable, or a product of two decision variables (e.g., $5x_1, 2x_1^2$, or $3x_1x_2$). Many algorithms have been developed for quadratic programming with the further assumption that the objective function is

concave. One applicable approach in this case is to use a direct extension of the simplex method to find the global maximum.

Convex programming covers a broad class of problems which include the assumptions that the objective function $f(\mathbf{x})$ is concave and all the functions $g_i(\mathbf{x})$ [for the constraints $g_i(\mathbf{x}) \leq b_i$] are convex functions, so that the feasible region is a convex set. These assumptions are enough to ensure that any local maximum is a global maximum.

Nonconvex programming includes all nonlinear programming models that do not satisfy the assumptions of convex programming. For these models, even if a local maximum is found, there is no guarantee that it will also be a global maximum. Except for a few special cases, there is no algorithm that will guarantee finding an optimal solution for such problems. A common approach for these problems is to apply an algorithm for finding a local maximum, and then restart it a number of times from a variety of initial trial solutions in order to find as many distinct local maxima as possible. The final step is to choose the best local maximum.

5.2 Solving Unconstrained Nonlinear Programming Models

Consider the problem of maximizing the objective function $f(\mathbf{x})$ over all possible values of $\mathbf{x}$. If the objective function is differentiable, then a necessary condition that a particular solution $\mathbf{x} = \mathbf{x}^*$ is optimal is

$$\frac{\partial f(\mathbf{x})}{\partial x_j} = 0 \text{ at } \mathbf{x} = \mathbf{x}^* \text{ for } j = 1, 2, \ldots, n. \tag{1.4}$$

If the objective function is also concave, then this is also a sufficient condition for optimality.

For these problems, one approach for finding an optimal solution is to apply a *gradient search procedure.* The *gradient* of $f(\mathbf{x})$, denoted $\nabla f(\mathbf{x})$, is the vector whose elements are the respective partial derivatives of $f(\mathbf{x})$. That is,

$$\nabla f(\mathbf{x}) = \left(\frac{\partial f(\mathbf{x})}{\partial x_1}, \frac{\partial f(\mathbf{x})}{\partial x_2}, \ldots, \frac{\partial f(\mathbf{x})}{\partial x_n} \right) \tag{1.5}$$

The significance of the gradient is that the (infinitesimal) change in $\mathbf{x}$ that maximizes the rate at which $f(\mathbf{x})$ increases is a move in the "direction" of the gradient $\nabla f(\mathbf{x})$. The gradient search procedure exploits this property. Each iteration involves changing the current trial solution $\mathbf{x}'$ as follows:

$$\text{Reset } \mathbf{x}' = \mathbf{x}' + t^* \nabla f(\mathbf{x}'), \tag{1.6}$$

where t^* is the positive value of t that maximizes $f(\mathbf{x}' + t\nabla f(\mathbf{x}'))$. The iterations of the gradient search procedure continue until $\nabla f(\mathbf{x}) = 0$ within a small tolerance.

If $f(\mathbf{x})$ is not a concave function, the gradient search procedure would converge to a local maximum, but not necessarily the global maximum. Several starting points can be tried in an attempt to find better local maxima.

Although the gradient search procedure is one popular (and particularly straightforward) search technique for unconstrained optimization, it is only a special case of a general class of such techniques called *iterative ascent methods* (or *iterative descent methods* when minimizing instead of maximizing). Each iteration of such a method begins by identifying a direction of ascent (called the *search direction*) from the current trial solution. A step is then taken in this direction from the current trial solution in order to find a new improved trial solution. These iterations are repeated until a test for convergence is satisfied.

The search direction for these methods typically is identified by obtaining a first- or second-order Taylor series expansion of the objective function around the current trial solution and then computing the direction that maximizes (or approximately maximizes) the expansion. The gradient search procedure uses a first-order expansion whereas *Newton's method* employs a second-order expansion. Although Newton's method thereby requires computing second derivatives, this provides a more rapid (quadratic) rate of convergence under favorable conditions. *Quasi-Newton methods* only compute approximations of the second derivatives.

For Newton-type methods, the length of the step to be taken in the search direction typically is determined in one of two ways. A *line search method* finds the steplength that maximizes (or at least approximately maximizes) the second-order Taylor series expansion along the line that leads from the current trial solution in the search direction. A *trust region method* only considers steplengths that are small enough that the second-order Taylor series expansion can be trusted to provide a reasonable approximation of the objective function.

5.3 Solving Constrained Nonlinear Programming Models

Now consider constrained optimization problems, as represented by (1.3). Necessary conditions for a given solution to be optimal are given by the *Karush-Kuhn-Tucker conditions* (also called *KKT conditions*). Assume that $f(\mathbf{x})$, $g_1(\mathbf{x})$, $g_2(\mathbf{x}), \ldots, g_m(\mathbf{x})$ are differentiable functions

satisfying certain regularity conditions. Then $\mathbf{x}^*$ can be an optimal solution if there exist numbers $u_1, u_2, \ldots, u_m$ such that all the following KKT conditions are satisfied:

$$\begin{aligned}
\frac{\partial f(\mathbf{x})}{\partial x_j} - \sum_{i=1}^{m} u_i \frac{\partial g_i(\mathbf{x})}{\partial x_j} &\leq 0 \qquad \text{at } \mathbf{x} = \mathbf{x}^*, \text{ for } j = 1, 2, \ldots, n. \\
x_j^* \left(\frac{\partial f(\mathbf{x})}{\partial x_j} - \sum_{i=1}^{m} u_i \frac{\partial g_i(\mathbf{x})}{\partial x_j} \right) &= 0 \qquad \text{at } \mathbf{x} = \mathbf{x}^*, \text{ for } j = 1, 2, \ldots, n. \\
g_i(\mathbf{x}^*) - b_i &\leq 0, \qquad \text{for } i = 1, 2, \ldots, m. \\
u_i[g_i(\mathbf{x}^*) - b_i] &= 0, \qquad \text{for } i = 1, 2, \ldots, m. \\
x_j^* &\geq 0, \qquad \text{for } j = 1, 2, \ldots, n. \\
u_i &\geq 0, \qquad \text{for } i = 1, 2, \ldots, m.
\end{aligned} \tag{1.7}$$

If the objective function is also concave and the feasible region forms a convex set (i.e. the model is a convex programming model), then this is also a *sufficient* condition for optimality.

There is no single standard algorithm that always is used to solve convex programming models. Many algorithms have been developed, each with its own advantages and disadvantages. These algorithms typically fall into one of three categories.

The first category is *gradient algorithms,* where the gradient search procedure is modified in some way to keep the search path from penetrating any constraint boundary. For example, one popular gradient method is the *generalized reduced gradient* (GRG) method.

The second category is *sequential unconstrained algorithms.* These algorithms often incorporate the constraints into either a penalty or barrier function that is subtracted from the objective function. When using a penalty function, the role of this function is to impose a penalty for violating constraints. With a barrier function, only feasible solutions are considered and then this function is used to impose a large penalty for approaching constraint boundaries. By subtracting a sequence of positive multiples of either kind of function from the objective function, the original constrained optimization problem is converted into a sequence of unconstrained optimization problems that can be solved by an unconstrained optimization algorithm, e.g., the gradient search procedure, inxxsequential unconstrained algorithms

For example, to solve (1.3), we can solve for $\mathbf{x}$ so as to

$$\text{Maximize } h(\mathbf{x}; r) = f(\mathbf{x}) - rB(\mathbf{x}) \tag{1.8}$$

where r is a positive constant and $B(\mathbf{x})$ *is* a positive barrier function that has the property that it is *small* when $\mathbf{x}$ is *far* from the boundary

of the feasible region, *large* when $\mathbf{x}$ is close to the boundary of the feasible region, and goes to infinity in the limit as the distance from the boundary of the feasible region approaches zero. The most common choice for $B(\mathbf{x})$ is

$$B(\mathbf{x}) = \sum_{i=1}^{m} \frac{1}{b_i - g_i(\mathbf{x})} + \sum_{j=1}^{n} \frac{1}{x_j} \tag{1.9}$$

By starting with a feasible initial trial solution that does not lie on the boundary of the feasible region and then applying an iterative ascent method that finds a series of improving trial solutions for (1.8), this barrier function forces all the trial solutions to remain within the interior of the feasible region for (1.3). However, an optimal solution for (1.3) might lie on or near the very boundary that the barrier function is preventing the search from approaching too closely. To alleviate this difficulty, a *sequence* of problems given by (1.8) are solved for successively smaller values of r, approaching zero, where the optimal solution obtained for each problem (or at least its approximation) is used as the initial trial solution for the next problem. The sequence of optimal solutions for the unconstrained problem (1.8) will converge to an optimal solution for the constrained problem (1.3) in the limit as r goes to zero, even if this latter solution lies on the boundary of the feasible region.

If a sequential unconstrained algorithm uses a *penalty function* instead of a barrier function, each of the unconstrained optimization problems considered has the form

$$\text{Maximize } h(\mathbf{x}; \rho) = f(\mathbf{x}) - \rho P(\mathbf{x}) \tag{1.10}$$

where ρ is a positive constant and $P(\mathbf{x})$ is a penalty function that penalizes constraint violations. Typically, $P(\mathbf{x})$ is an additive function with one nonnegative term for each of the constraints in the original nonlinear programming problem, where this term is *zero* if the corresponding constraint is satisfied, *small* if the constraint is violated but barely so, *large* if the constraint is substantially violated, and grows without bound as the size of the violation increases. For example, consider the case of a nonlinear programming problem where the only constraints are functional constraints in equality form,

$$g_i(\mathbf{x}) = b_i, \text{ for } i = 1, 2, \ldots, m. \tag{1.11}$$

A commonly used penalty function for this case is the quadratic function

$$P(\mathbf{x}) = \sum_{i=1}^{m} (b_i - g_i(\mathbf{x}))^2. \tag{1.12}$$

With a functional constraint in inequality form, $g_i(\mathbf{x}) \leq b_i$, the same quadratic term would be included in $P(\mathbf{x})$ *if* the constraint is violated, so that the term would be set to zero if $b_i - g_i(\mathbf{x}) \geq 0$. Similarly, with nonnegativity constraints, $x_j \geq 0$ for $j = 1, 2, \ldots, n$, quadratic terms x_j^2 would be included in $P(\mathbf{x})$ *if* $x_j \geq 0$ is violated, so that the term would be set to zero if $x_j \geq 0$. Starting with a trial solution that violates one or more of the constraints, an iterative ascent method is used to solve a sequence of unconstrained optimization problems of the form (1.10) for successively larger values of ρ (where the final trial solution from each problem becomes the initial trial solution for the next one). By using a sequence of values of ρ that would go to infinity in the limit, the sequence of optimal solutions for the unconstrained optimization problems converges to a solution that is both feasible and optimal for the original nonlinear programming problem.

The third category of algorithms for convex programming is *sequential approximation algorithms,* including linear approximation and quadratic approximation methods. These algorithms replace a nonlinear objective function by a sequence of linear or quadratic approximations. These algorithms are particularly well suited to linearly constrained nonlinear programming models, where these approximations allow repeated application of efficient linear programming or quadratic programming algorithms.

For example, consider a linearly constrained nonlinear programming model. At any given trial solution $\mathbf{x}'$, the objective function can be approximated by the first-order Taylor series expansion of $f(\mathbf{x})$ around $\mathbf{x} = \mathbf{x}'$. That is,

$$f(\mathbf{x}) \approx f(\mathbf{x}') + \sum_{j=1}^{n} \frac{\partial f(\mathbf{x}')}{\partial x_j}(x_j - x_j'). \tag{1.13}$$

Using this linear approximation of the objective function, a linear programming algorithm can be used to find an optimal solution for the resulting linear programming model. By considering the line segment between $\mathbf{x}'$ and this optimal solution, a line search can be conducted to maximize the original objective function $f(\mathbf{x})$ over this line segment to find a new trial solution. A new approximation for the objective function is then derived at the new trial solution. The procedure is repeated until it converges to a solution.

Algorithms for nonconvex nonlinear programming models and for nonlinear programming models with nondifferentiable functions are an area of ongoing research. However, these are very difficult problems. These

appear to be promising areas for the application of evolutionary optimization algorithms.

6. Simulation

The preceding sections have focused on decision making when the consequences of alternative decisions are known with a reasonable degree of certainty. This decision-making environment enabled formulating helpful models with objective functions that specify the estimated consequences of any combination of decisions. Although these consequences usually cannot be predicted with complete certainty, they could at least be estimated with enough accuracy to justify using such models (along with sensitivity analysis, etc.).

However, decisions often must be made in environments that are much more fraught with uncertainty. Furthermore, the decisions may need to take into account uncertainty about many future events. This is the case when making decisions about how to design and operate stochastic systems (systems that evolve over time in a probabilistic manner) so as to optimize their performance.

Simulation is a widely used technique for analyzing stochastic systems in preparation for making these kinds of decisions. This technique involves using a computer to *imitate* (simulate) the operation of an entire process or system. For example, simulation is frequently used to perform risk analysis on financial processes by repeatedly imitating the evolution of the transactions involved to generate a profile of the possible outcomes. Simulation also is widely used to analyze stochastic systems that will continue operating indefinitely. For such systems, the computer randomly generates and records the occurrences of the various events that drive the system just as if it were physically operating. Because of its speed, the computer can simulate even years of operations in a matter of seconds. Recording the performance of the simulated operation of the system for a number of alternative designs or operating procedures then enables evaluating and comparing these alternatives before choosing one.

The technique of simulation has long been an important tool of the designer. For example, simulating airplane flight in a wind tunnel is standard practice when a new airplane is designed. Theoretically, the laws of physics could be used to obtain the same information about how the performance of the airplane changes as design parameters are altered. However, as a practical matter, the analysis would be too complicated to do it all. Another alternative would be to build real airplanes with alternative designs and test them in actual flight to choose the final

design, but this would be far too expensive (as well as unsafe). Therefore, after some preliminary theoretical analysis is performed to develop a rough design, simulating flight in a wind tunnel is a vital tool for experimenting with specific designs. This simulation amounts to imitating the performance of a real airplane in a controlled environment in order to *estimate* what its actual performance will be. After a detailed design is developed in this way, a prototype model can be built and tested in actual flight to fine-tune the final design.

Simulation plays essentially this same role in many OR/MS studies. However, rather than designing an airplane, the OR/MS team is concerned with developing a design or operating procedure for some stochastic system. Rather than use a wind tunnel, the performance of the real system is imitated by using probability distributions to randomly generate various events that occur in the system. Therefore, a simulation model synthesizes the system by building it up component by component and event by event. Then the model runs the simulated system to obtain statistical observations of the performance of the system that result from various randomly generated events. Because the simulation runs typically require generating and processing a vast amount of data, these simulated statistical experiments are inevitably performed on a computer.

When simulation is used as part of an OR/MS study, commonly it is preceded and followed by the same steps described earlier for the design of an airplane. In particular, some preliminary analysis is done first (perhaps with approximate mathematical models) to develop a rough design of the system (including its operating procedures). Then simulation is used to experiment with specific designs to estimate how well each will perform. After a detailed design is developed and selected in this way, the system probably is tested in actual use to fine-tune the final design.

To prepare for simulating a complex system, a detailed *simulation model* needs to be formulated to describe the operation of the system and how it is to be simulated. A simulation model has several basic building blocks:

- A definition of the *state of the system.*
- A list of the *possible states* of the system that can occur.
- A list of the *possible events* that would change the state of the system.
- A provision for a *simulation clock,* located at some address in the simulation program, that will record the passage of (simulated) time.

- A method for *randomly generating the events* of the various kinds.
- A formula for identifying *state transitions* that are generated by the various kinds of events.

Great progress is being made in developing special software for efficiently integrating the simulation model into a computer program. This software includes general-purpose simulation languages, applications-oriented simulators for simulating specific types of systems, and animation software for displaying computer simulations in action, as well as software for performing simulations on spreadsheets.

Two broad categories of simulations are discrete-event and continuous simulations. A *discrete-event simulation* is one where changes in the state of the system occur instantaneously at random points in time as a result of the occurrence of discrete events. A *continuous simulation* is one where changes in the state of the system occur continuously over time.

Simulation now is one of the most widely used OR/MS techniques, and it is continuing to grow in popularity because of its great versatility. We list below some examples of important types of applications of simulation.

- Design and operation of queueing systems
- Managing inventory systems
- Estimating the probability of completing a project by the deadline
- Design and operation of manufacturing systems
- Design and operation of distribution systems
- Financial risk analysis
- Health care applications
- Applications to other service industries

Simulation is a powerful tool for analyzing stochastic systems such as these by providing estimates of how the system would perform with various alternative designs and operating procedures. However, simulation does not determine by itself how to optimize the performance of the system. A supplementary technique is needed to use the estimates provided by simulation to search for the optimal values (or at least the approximately optimal values) of the decision variables involving the design and operating procedure for the system. Evolutionary optimization algorithms (and other metaheuristics) have a strong potential for being used in conjunction with simulation in this way.

7. Further Reading

All the topics covered in this chapter are developed in much greater detail in Hillier and Lieberman (2001). Hillier et al. (2000) also expand on these topics with a relatively applied orientation.

In addition, Hillier and Lieberman (2001) provide a number of selected references for each of these topics. We will mention here just one book that is devoted to each topic.

Vanderbei (2001) focuses on linear programming. Schneiderjans (1995) deals with goal programming. Nemhauser and Wolsey (1988) provide a treatise on integer programming. Bertsekas (1995) focuses on nonlinear programming. Fishman (1996) provides a leading reference on simulation.

References

Bertsekas, D.P. (1995). *Nonlinear Programming,* Athena Scientific, Belmont, MA.

Fishman, G.S. (1996). *Monte Carlo: Concepts, Algorithms and Applications,* Springer-Verlag, New York.

Hillier, F.S., M.S. Hillier, and G.J. Lieberman (2000). *Introduction to Management Science: A Modeling and Case Studies Approach with Spreadsheets,* Irwin/McGraw-Hill, Burr Ridge, IL.

Hillier, F.S., and G.J. Lieberman (2001). *Introduction to Operations Research,* 7th ed., McGraw-Hill, Burr Ridge, IL.

Nemhauser, G.L., and L.A. Wolsey (1988). *Integer and Combinatorial Optimization.* Wiley, New York. (Reprinted in 1999.)

Schneiderjans, M. (1995). *Goal Programming: Methodology and Applications,* Kluwer Academic Publishers, Boston.

Vanderbei, R.J. (2001). *Linear Programming: Foundations and Extensions,* 2nd ed., Kluwer Academic Publishers, Boston.

This page intentionally left blank

Chapter 2

EVOLUTIONARY COMPUTATION

A Gentle Introduction

Xin Yao

Abstract This chapter gives a gentle introduction to evolutionary computation, a field in which evolutionary optimisation is one of the most important research areas. Unlike most introductions to evolutionary computation which are based on its simplified biological link, this chapter emphasises the link between evolutionary computation and artificial intelligence and computer science. In fact, this whole book is centred around problem-solving, e.g., optimisation, using evolutionary computation techniques. It does not deal with the issue of biological modelling.

Keywords: Evolutionary computation, global optimisation, combinatorial optimisation, evolutionary learning, evolutionary design.

1. What Is Evolutionary Computation

Evolutionary computation is the study of computational systems which use ideas and get inspirations from natural evolution and adaptation. It aims at understanding such computational systems and developing more robust and efficient ones for solving complex real-world problems. The problems dealt with by such computational systems are usually highly nonlinear and contain inaccurate and noisy data.

Traditional computational systems are good at accurate and exact computation but brittle. They are not designed for processing inaccurate, noisy and complex data although they might excel at dealing with complicated data. For example, the classical simplex method is an invaluable mathematical programming technique which has been applied to numerous practical problems successfully. However, it requires a problem to be formulated in exact and accurate mathematical forms. It does not work well for problems where the objective function cannot be expressed mathematically, is noisy, and changes with time. Evolu-

tionary computation is a field where such problems will be studied in depth. It complements the study of traditional computational systems.

Many evolutionary computation techniques get their ideas and inspirations from molecular evolution, population genetics, immunology, etc. Some of the terminologies used in evolutionary computation have been borrowed from these fields to reflect their connections, such as *genetic* algorithms, *genotypes, phenotypes, species,* etc. Although the research in evolutionary computation could help us understand some biological phenomena better, its primary aim is not to build biologically plausible models. There is no requirement in evolutionary computation that a technique developed must be biologically plausible. The primary aim is to study and develop robust and efficient computational systems for solving complex real-world problems.

Evolutionary computation is an emerging field which has grown rapidly in recent years. There are at least two international journals which are dedicated to this field: *IEEE Transactions on Evolutionary Computation* and *Evolutionary Computation* (MIT Press). Other journals which have a large evolutionary computation component include *IEEE Transactions on Systems, Man, and Cybernetics* and *BioSystems* (Elsevier). There are also many international conferences on evolutionary computation held each year, such as the annual *IEEE International Conference on Evolutionary Computation, Evolutionary Programming Conference* and *Genetic Programming Conference,* and bi-annual *International Conference on Genetic Algorithms, International Conference on Parallel Problem Solving from Nature,* and *Asia-Pacific Conference on Simulated Evolution and Learning.*

1.1 A Brief History

Evolutionary computation encompasses several major branches, i.e., evolution strategies, evolutionary programming, genetic algorithms and genetic programming, due largely to historical reasons. At the philosophical level, they differ mainly in the level at which they simulate evolution. At the algorithmic level, they differ mainly in their representations of potential solutions and their operators used to modify the solutions. From a computational point of view, representation and search are two key issues. This book will look at their differences more at the algorithmic level than at the philosophical level.

Evolution strategies were first proposed by Rechenberg and Schwefel in 1965 as a numerical optimisation technique. The original evolution strategy did not use populations. A population was introduced into evolution strategies later (Schwefel, 1981; Schwefel, 1995).

Evolutionary programming was first proposed by Fogel *et al.* in mid 1960's as one way to achieve artificial intelligence (Fogel et al., 1966). Several examples of evolving finite state machines were demonstrated (Fogel et al., 1966). Since late 1980's, evolutionary programming was also applied to various combinatorial and numerical optimisation problems.

The current framework of genetic algorithms was first proposed by Holland (Holland, 1975) and his students (Jong, 1975) in 1975 although some of the ideas appeared as early as 1957 in the context of simulating genetic systems (Fraser, 1957). Genetic algorithms were first proposed as adaptive search algorithms, although they have mostly been used as a global optimisation algorithm for either combinatorial or numerical problems. They are probably the most well-known branch of evolutionary computation.

A special sub-branch of genetic algorithms is . Genetic programming can be regarded as an application of genetic algorithms to evolve tree-structured chromosomes. Historically, those trees represent LISP programs. The term of genetic programming was first used by Koza in the above sense (Koza, 1989; Koza, 1990). de Garis used the term of genetic programming to mean a quite different thing. He regarded genetic programming as the genetic evolution of artificial neural networks (de Garis, 1990). This book will follow Koza's explanation of genetic programming since de Garis is no longer using the term.

In recent years, a general term of evolutionary algorithms has been used by more and more researchers to include all three major algorithms, i.e., evolution strategies, evolutionary programming and genetic algorithms, since they use almost the same computational framework. This is the view taken by this book.

1.2 A General Framework of Evolutionary Algorithms

All evolutionary algorithms have two prominent features which distinguish themselves from other search algorithms. First, they are all population-based. Second, there is communications and information exchange among individuals in a population. Such communications and information exchange are the result of selection and/or recombination in evolutionary algorithms. A general framework of evolutionary algorithms can be summarised by Figure 2.1, where the search operators are also called genetic operators for genetic algorithms. They are used to generate offspring (new individuals) from parents (existing individuals).

1 Set $i = 0$;

2 Generate the initial **population** $P(i)$ at random;

3 REPEAT

(a) Evaluate the fitness of each individual in $P(i)$;

(b) Select parents from $P(i)$ based on their fitness;

(c) Apply **search operators** to the parents and produce generation $P(i+1)$;

4 UNTIL the population converges or the maximum time is reached

Figure 2.1. A General Framework of Evolutionary Algorithms.

Obviously Figure 2.1 specifies a whole class of algorithms, not any particular ones. Different representations of individuals and different schemes for implementing fitness evaluation, selection and search operators define different algorithms.

1.3 Evolution Strategies

For evolution strategies (Schwefel, 1995; Bäck, 1996), the representation of individuals is often very close to a problem's natural representation. It does not emphasise the genetic representation of individuals. For example, an individual is represented as a vector of real numbers rather than a binary string for numerical optimisation problems. Evolution strategies usually use a deterministic selection scheme, Gaussian mutation, and discrete or intermediate recombination. The term crossover is seldom used in the context of evolution strategies because evolution strategies do not simulate evolution at the genetic level.

There are two major deterministic selection schemes in evolution strategies (Schwefel, 1981; Schwefel, 1995), i.e., $(\lambda + \mu)$ and (λ, μ) where μ is the population size (which is the same as the number of parents) and λ the number of offspring generated from all μ parents. In $(\lambda + \mu)$ evolution strategies, λ offspring will be generated from μ parents. The μ fittest individuals from $\lambda + \mu$ candidates will be selected to form the next generation. In (λ, μ) evolution strategies, the μ fittest individuals from λ offspring only will be selected to form the next generation. As a result, $\lambda \geq \mu$ is required.

Mutation in evolution strategies is often implemented by adding a Gaussian random number to a parent. Assume $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ is a parent (individual), then an offspring will be generated by mutation as follows:

$$x_i' = x_i + N_i(0, \sigma_i) \tag{2.1}$$

where $N_i(0, \sigma_i)$ is a normally distributed random number with mean 0 and standard deviation σ_i. The n random numbers are generated independently.

One important parameter in the Gaussian mutation is the standard deviation, σ_i. Its selection is quite important in determining the performance of evolution strategies. Unfortunately, its optimal value is problem dependent as well as dimension dependent. Schwefel (Schwefel, 1981) proposed to include σ_i's as part of an individual so that it can be evolved automatically. This is often called *self-adaptation* in evolution strategies. It is one of the major differences between evolution strategies and genetic algorithms. In many implementations, σ_i's will be mutated first, and then x_i is mutated using the new σ_i'.

Mutating different components of an vector independently may not be appropriate for some problems because those components may not be independent at all. To address this issue, co-variance has been introduced as another additional part of an individual. It is unclear at this stage whether such self-adaptation is beneficial for most problem as the search space will be increased exponentially as we triple (at least) the individual size. Further work will be necessary in this area.

Recombination in evolution strategies takes two major forms, i.e., discrete and intermediate recombinations. Discrete recombination mixes components of two parent vectors. For example, given two parents $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ and $\mathbf{y} = (y_1, y_2, \ldots, y_n)$. The offspring $\mathbf{x}' = (x_1', x_2', \ldots, x_n')$ and $\mathbf{y}' = (y_1', y_2', \ldots, y_n')$ can be generated as follows:

$$x_i' = \begin{cases} x_i & \text{with probability } p_{recombination} \\ y_i & \text{otherwise} \end{cases}$$

$\mathbf{y}'$ will be the complement of $\mathbf{x}'$.

Intermediate recombination is usually based on some kind of averaging. For example, given two parents $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ and $\mathbf{y} = (y_1, y_2, \ldots, y_n)$. The offspring $\mathbf{x}' = (x_1', x_2', \ldots, x_n')$ and $\mathbf{y}' = (y_1', y_2', \ldots, y_n')$ can be generated as follows:

$$x_i' = x_i + \alpha(y_i - x_i)$$

where α is a weighting parameter in (0,1). It is traditionally set to 0.5. It can also be generated at random. $\mathbf{y}'$ can be generated similarly.

According to the description by Bäck and Schwefel (Bäck and Schwefel, 1993), a (μ, λ) evolution strategy can be implemented as follows:

1 Generate the initial population of μ individuals, and set $k = 1$. Each individual is taken as a pair of real-valued vectors, $(\mathbf{x}_i, \eta_i)$, $\forall i \in \{1, \cdots, \mu\}$, where η plays the role of σ (i.e., the standard deviation).

2 Evaluate the fitness value for each individual $(\mathbf{x}_i, \eta_i)$, $\forall i \in \{1, \cdots, \mu\}$, of the population.

3 Each parent $(\mathbf{x}_i, \eta_i)$, $i = 1, \cdots, \mu$, creates λ/μ offspring on average, so that a total of λ offspring are generated: for $i = 1, \cdots, \mu$, $j = 1, \cdots, n$, and $k = 1, \cdots, \lambda$,

$$\eta_k'(j) = \eta_i(j) \exp(\tau' N(0,1) + \tau N_j(0,1)) \quad (2.2)$$

$$\mathbf{x}_k'(j) = \mathbf{x}_i(j) + \eta_i'(j) N_j(0,1) \quad (2.3)$$

where $\mathbf{x}_i(j)$, $\mathbf{x}_k'(j)$, $\eta_i(j)$ and $\eta_k'(j)$ denote the j-th component of the vectors $\mathbf{x}_i$, $\mathbf{x}_k'$, η_i and η_k', respectively. $N(0,1)$ denotes a normally distributed one-dimensional random number with mean zero and standard deviation one. $N_j(0,1)$ indicates that the random number is generated anew for each value of j. The factors τ and τ' are usually set to $\left(\sqrt{2\sqrt{n}}\right)^{-1}$ and $\left(\sqrt{2n}\right)^{-1}$ (Bäck and Schwefel, 1993).

4 Evaluate the fitness of each offspring $(\mathbf{x}_i', \eta_i')$, $\forall i \in \{1, \cdots, \lambda\}$.

5 Sort offspring $(\mathbf{x}_i', \eta_i')$, $\forall i \in \{1, \cdots, \lambda\}$ into a non-descending order according to their fitness values, and select the μ best offspring out of λ to be parents of the next generation.

6 Stop if the stopping criterion is satisfied; otherwise, $k = k + 1$ and go to Step 3.

1.4 Evolutionary Programming

When used for numerical optimisation, evolutionary programming (Fogel et al., 1966; Fogel, 1991; Fogel, 1995) is very similar to evolution strategies in terms of algorithm. It uses vectors of real numbers as individuals, Gaussian mutation and self-adaptation as described above. The most noticeable differences between evolutionary programming and evolution strategies are recombination and selection. Evolutionary programming does not use any recombination or crossover, but uses a probabilistic competition (i.e., a kind of tournament selection) as the selection mechanism. Of course, there is no reason why evolutionary programming

cannot have recombination and why evolution strategies cannot have a probabilistic selection scheme from the algorithmic point of view.

The origins of evolutionary programming and evolution strategies are quite different. Evolutionary programming was first proposed to simulate intelligence by evolving finite state machines, while evolution strategies were proposed to optimise numerical parameters. It was unclear how recombination could be usefully applied to finite state machines.

According to the description by Bäck and Schwefel (Bäck and Schwefel, 1993), evolutionary programming can be implemented as follows:

1 Generate the initial population of μ individuals, and set $k = 1$. Each individual is taken as a pair of real-valued vectors, $(\mathbf{x}_i, \eta_i)$, $\forall i \in \{1, \cdots, \mu\}$, where $\mathbf{x}_i$'s are objective variables and η_i's are standard deviations for Gaussian mutations.

2 Evaluate the fitness score for each individual $(\mathbf{x}_i, \eta_i)$, $\forall i \in \{1, \cdots, \mu\}$, of the population.

3 Each parent $(\mathbf{x}_i, \eta_i)$, $i = 1, \cdots, \mu$, creates a single offspring $(\mathbf{x}_i', \eta_i')$ by: for $j = 1, \cdots, n$,

$$\eta_i'(j) = \eta_i(j) \exp(\tau' N(0,1) + \tau N_j(0,1)) \quad (2.4)$$

$$x_i'(j) = x_i(j) + \eta_i'(j) N_j(0,1), \quad (2.5)$$

where $x_i(j)$, $x_i'(j)$, $\eta_i(j)$ and $\eta_i'(j)$ denote the j-th component of the vectors $\mathbf{x}_i$, $\mathbf{x}_i'$, η_i and η_i', respectively. $N(0,1)$ denotes a normally distributed one-dimensional random number with mean 0 and standard deviation 1. $N_j(0,1)$ indicates that the random number is generated anew for each value of j. The factors τ and τ' have commonly set to $\left(\sqrt{2\sqrt{n}}\right)^{-1}$ and $\left(\sqrt{2n}\right)^{-1}$ (Bäck and Schwefel, 1993; Fogel, 1994).

4 Calculate the fitness of each offspring $(\mathbf{x}_i', \eta_i')$, $\forall i \in \{1, \cdots, \mu\}$.

5 Conduct pairwise comparison over the union of parents $(\mathbf{x}_i, \eta_i)$ and offspring $(\mathbf{x}_i', \eta_i')$, $\forall i \in \{1, \cdots, \mu\}$. For each individual, q opponents are chosen uniformly at random from all the parents and offspring. For each comparison, if the individual's fitness is no smaller than the opponent's, it receives a "win."

6 Select the μ individuals out of $(\mathbf{x}_i, \eta_i)$ and $(\mathbf{x}_i', \eta_i')$, $\forall i \in \{1, \cdots, \mu\}$, that have the most wins to be parents of the next generation.

7 Stop if the halting criterion is satisfied; otherwise, $k = k + 1$ and go to Step 3.

1.5 Genetic Algorithms

Genetic algorithms (Holland, 1975; Goldberg, 1989; Michalewicz, 1996) are quite different from evolution strategies and evolutionary programming in terms of individual representation and search operators. Genetic algorithms emphasise genetic encoding of potential solutions into chromosomes and apply genetic operators to these chromosomes. This is equivalent to transforming the original problem from one space to another space. It is obvious that the genetic representation will be crucial to the success of genetic algorithms. A good representation will make a problem easier to solve. A poor representation will do the opposite. The issues faced by genetic algorithms in general are the same as those which have haunted many artificial intelligence problems for years, i.e., representation and search. In other words, a crucial issue in applying genetic algorithms to a problem is how to find a representation which can be searched efficiently.

A canonical genetic algorithm (also called simple genetic algorithm sometimes) (Goldberg, 1989) is the one which uses binary representation, one point crossover and bit-flipping mutation. Binary representation means that each individual will be represented by a number of binary bits, 0 or 1. One point crossover is carried out as follows: Given two binary strings, x and y, of length n. Generate a crossover point between 1 and $n-1$ (inclusively) uniformly at random, say r. Then the first offspring consists of the first r bits of x and the last $n-r$ bits of y. The second offspring consists of the first r bits of y and the last $n-r$ bits of x. Mutation is carried out bit-wise. That is, every bit of an individual has certain probability of being flipped from 0 to 1 or from 1 to 0. A canonical genetic algorithm can be implemented as follows:

1 Generate the initial population $P(0)$ at random and set $i = 0$;

2 REPEAT

(a) Evaluate the fitness of each individual in $P(i)$.

(b) Select parents from $P(i)$ based on their fitness as follows: Given the fitness of n individuals as $f_1, f_2, \ldots, f_n$. Then select individual i with probability

$$p_i = \frac{f_i}{\sum_{j=1}^{n} f_j}.$$

This is often called roulette wheel selection of fitness proportional selection.

(c) Apply crossover to selected parents;

(d) Apply mutation to crossed-over new individuals;

(e) Replace parents by the offspring to produce generation $P(i+1)$;

3 UNTIL the halting criterion is satisfied

1.6 Other Topics in Evolutionary Computation

There are numerous variants of classical evolution strategies, evolutionary programming and genetic algorithms described above. Some of the evolutionary algorithms can hardly be classified into any of these three categories. Evolutionary computation includes much more than just three kinds of algorithms. It also covers topics such as artificial immune systems, artificial ecological systems, co-evolutionary systems, evolvable hardware, self-adaptive systems, etc.

2. A Brief Overview of Evolutionary Computation

The current research and development in evolutionary computation can be classified into three major areas, i.e., evolutionary computation theory, evolutionary optimisation and evolutionary learning. There are, of course, overlaps among these areas.

2.1 Evolutionary Computation Theory

The theoretical work in evolutionary computation has concentrated on three main topics. The first one is the theoretical analysis of convergence and convergence rate of evolutionary algorithms. There has been some work on the convergence and convergence rate of evolution strategies (Schwefel, 1981; Schwefel, 1995; Beyer, 1994), evolutionary programming (Fogel, 1992; Fogel, 1995) and genetic algorithms (Eiben et al., 1991; Rudolph, 1994; Rudolph, 1996; Reynolds and Gomatam, 1996; Suzuki, 1993; Suzuki, 1995; Rudolph, 1994a). They are very general results which describe the asymptotic behaviour of certain class of evolutionary algorithms under different conditions. However, few of them studied the relationship between the convergence rate and the problem size.

The second main topic in evolutionary computation theory is the study of problem hardness with respect to evolutionary algorithms. That is, the aim is to investigate what kind of problems is hard for evolutionary algorithms and what is easy for them. If we knew the characteristics of a problem which make evolutionary algorithms hard or easy to solve, we would be able to better understand how and when evolutionary algorithms would work. This will be of enormous practical value in addition

to of theoretical interest. Centered around this topic, there was some work on genetic algorithm deceptive problems (Goldberg, 1989; Liepins and Vose, 1991; Deb and Goldberg, 1993). Such work tried to characterise problems that are hard for genetic algorithms to solve as deceptive. It has been pointed out that this approach is rather problematic (Grefenstette, 1993). There have been other approaches to the understanding of what make a problem hard for genetic algorithms, such as building block and schema analysis (Holland, 1975; Goldberg, 1989; Davidor, 1991), Walsh analysi based on Walsh functions (Heckendorn and Whitley, 1997), fitness landscape analysis (Manderick et al., 1991; Hordijk, 1996) and fitness distance correlation (Jones, 1995), etc. All these approaches have made certain progress towards a better understanding of how genetic algorithms work, but there is still a long way to go to gain a full understanding of how and when a genetic algorithm would work.

The third main topic in evolutionary computation theory is computational complexity of evolutionary algorithms. This is one of the most important research topics where little progress has been made. Evolutionary algorithms have been used extensively in both combinatorial and numerical optimisation in spite of the original emphasis on search and adaptation (Fogel et al., 1966; Holland, 1975; Holland, 1992; DeJong, 1993). There are established algorithms and their complexity results for many of these optimisation problems. However, it is unclear whether evolutionary algorithms can perform any better than other approximate or heuristic algorithms in terms of worst or average time complexity. There has not been any concrete result on the computational complexity of an evolutionary algorithm on a nontrivial problem, especially a combinatorial problem, although the complexity theory is well established for combinatorial decision and optimisation problems (Garey and Johnson, 1979).

2.2 Evolutionary Optimisation

Evolutionary optimisation is probably the most active and productive area in evolutionary computation measured by the number of papers published and the number of successful applications reported. Although neither evolutionary programming nor genetic algorithms were first proposed as optimisation algorithms, people had quickly realised they could adapt these algorithms to carry out combinatorial and function optimisation. Hence a flood of variants of classical algorithms were proposed and applied to different optimisation problems.

So far most of the evolutionary optimisation work belongs to numerical optimisation. Both constrained (Michalewicz and Schoenauer,

1996; Kim and Myung, 1997) and unconstrained (Yao and Liu, 1996; Yao and Liu, 1997) numerical optimisation has been studied. There has also been research on multiobjective optimisation by evolutionary algorithms (Fonseca and Fleming, 1995; Fonseca and Fleming, 1998; Fonseca and Fleming, 1998a).

When genetic algorithms are applied to numerical function optimisation, vectors of real numbers are usually encoded into binary bit strings. Different binary encoding methods have been proposed, such as Gray coding (Goldberg, 1989) and delta coding (Mathias and Whitley, 1994). Delta coding actually changes the representation during search. In spite of all these efforts in finding the best encoding method for real numbers, it is still unclear whether it is necessary to transform real numbers into binery strings.

Evolution strategies and evolutionary programming use vectors of real numbers directly as individuals and thus avoid the burden of finding a suitable encoding method for individuals. There have been some comparative studies between the binary representation used by genetic algorithms and the real representation used by evolutionary programming (Fogel and Atmar, 1990; Fogel, 1995a). However, more extensive comparisons need to be carried out to test the performance of different algorithms and find out why an algorithm performs well (or poorly) for certain problems.

In addition to numerical optimisation, evolutionary algorithms have also been used to tackle various combinatorial optimisation problems, such as the travelling salesman problem (Grefenstette et al., 1985; Fogel, 1988; Yao, 1993), transportation problem (Michalewicz, 1992; Vignaux and Michalewicz, 1991), switchbox routing in integrated circuits (Lienig and Thulasiraman, 1995), cutting stock problem (Hinterding and Khan, 1995; Liang et al., 1998), lecture room assignment problem (Luan and Yao, 1994), etc. Some of these results are quite competitive in comparison with more traditional approaches. In particular, hybrid algorithms which combine evolutionary algorithms with others (such as simulated annealing (Yao, 1991) and local search methods (Kido et al., 1994)) have shown a lot of promises in dealing with hard combinatorial optimisation problems.

2.3 Evolutionary Learning

Evolutionary learning includes many topics, such as learning classifier systems, evolutionary artificial neural networks, co-evolutionary learning, self-adaptive systems, etc. The primary goal of evolutionary learning is the same as that of machine learning in general. Evolution-

ary learning can be regarded as the evolutionary approach to machine learning. It has been used in the framework of supervised learning, reinforcement learning and unsupervised learning, although it appears to be most promising as a reinforcement learning method.

2.3.1 Learning Classifier Systems. Learning classifier systems (Holland, 1986; Holland, 1988), also known as classifier systems, are probably the oldest and best known evolutionary learning systems, although they did not work very well in their classical form (Westerdale, 1997). Some of the recent systems have improved this situation (Wilson, 1995; Colombetti and Dorigo, 1998). Due to its historical importance, a brief introduction to the classical learning classifier system (Holland, 1986; Holland, 1988) will be presented here.

Learning classifier systems are a particular class of message-passing, rule-based systems (Holland, 1988). They can also be regarded as a type of adaptive expert system that uses a knowledge base of production rules in a low-level syntax that can be manipulated by a genetic algorithm (Smith and Goldberg, 1992). In a classifier system, each low-level rule is called a *classifier.*

A genetic algorithm is used in classifier systems to discover new classifiers by crossover and mutation. The strength of a classifier is used as its fitness. The genetic algorithm is only applied to the classifiers after certain number of operational cycles in order to approximate strengths better. There are two approaches to classifier systems; *the Michigan approach* and *the Pitts approach.* For the Michigan approach, each individual in a population is a classifier. The whole population represents a complete classifier system. For the Pitts approach, each individual in a population represents a complete classifier system. The whole population includes a number of competing classifier systems.

2.3.2 Evolutionary Artificial Neural Networks. Evolutionary artificial neural networks can be considered as a combination of artificial neural networks and evolutionary algorithms (Yao, 1991b; Yao, 1993a; Yao, 1995a). Evolutionary algorithms have been introduced into artificial neural networks at three different levels: the evolution of connection weights, architectures, and learning rules (Yao, 1993a; Yao, 1995a). At present, most work on evolutionary artificial neural networks concentrates on the evolution of architectures, i.e., connectivities of ANNs (Yao and Shi, 1995; Liu and Yao, 1996; Yao and Liu, 1997a; Yao and Liu, 1998). Very good results have been achieved for some artificial and real-world benchmark problems.

One of the most important benefits of evolutionary artificial neural networks is that a near optimal (in terms of generalisation) artificial neural network with both structure and weights can be evolve automatically without going through a tedious trial-and-error manual design process. The results obtained so far have demonstrated that very compact artificial neural networks with good generalisation can be evolved (Yao and Liu, 1997a).

2.3.3 Co-evolutionary Learning. "Coevolution refers to the simultaneous evolution of two or more species with coupled fitness." (Rosin and Belew, 1997) Co-evolutionary learning has two different forms. In the first form, two or more populations are evolved at the same time (Hillis, 1991). The fitness of an individual in one population depends on the individuals in another population. There is no crossover or other information exchange between two populations. This can be regarded as co-evolution at the population level.

The second form of co-evolution is at the individual level. There is only one population involved. The fitness of an individual in the population depends on other individuals in the same population (Axelrod, 1987; Yao and Darwen, 1994; Darwen and Yao, 1995; Darwen and Yao, 1996; Darwen and Yao, 1997). For example, the same strategy for playing an iterated prisoner's dilemma game may get quite different fitness values depending on what other strategies are in the same population. Both forms of co-evolution have a dynamic environment and a dynamic fitness function. This is an active area of research.

3. Evolutionary Algorithm and Generate-and-Test Search Algorithm

Although evolutionary algorithms are often introduced from the point of view of *survival of the fittest* and from the analogy to natural evolution, they can also be understood through the framework of **generate-and-test** search. The advantage of introducing evolutionary algorithms as a type of generate-and-test search algorithms is that the relationships between evolutionary algorithms and other search algorithms, such as simulated annealing (Kirkpatrick et al., 1983; Szu and Hartley, 1987; Yao, 1991a; Yao, 1995), tabu search (Glover, 1989; Glover, 1990), etc., can be made clearer and thus easier to explore. Under the framework of generate-and-test search, different search algorithms investigated in artificial intelligence, operations research, computer science, and evolutionary computation can be unified together. Such interdisciplinary studies are expected to generate more insights into search algorithms in general.

A general framework of generate-and-test search can be described by Figure 2.2.

1 Generate the initial solution at random and denote it as the current solution;

2 **Generate** the next solution from the current one by *perturbation*;

3 **Test** whether the newly generated solution is *acceptable*;

(a) Accepted it as the current solution if yes;

(b) Keep the current solution unchanged otherwise.

4 Goto Step 2 if the current solution is not satisfactory, stop otherwise.

Figure 2.2. A General Framework of Generate-and-Test.

It is quite clear that various hill-climbing algorithms can be described by Figure 2.2 with different strategies for perturbation. They all require the new solution to be no worse than the current one to be acceptable. simulated annealing does not have such a requirement. It regards a worse solution to be acceptable with certain probability. The difference among classical simulated annealing (Kirkpatrick et al., 1983), fast simulated annealing (Szu and Hartley, 1987), very fast simulated annealing (Ingber, 1989), and a new simulated annealing (Yao, 1995) lies in the difference in their perturbations, i.e., methods of generating the next solution.

Evolutionary algorithms can be regarded as a population-based version of generate-and-test search. They use search operators like crossover and mutation to *generate* new solutions, and use selection to *test* which solutions are fitter than others. From this point of view, it is clear that we do not have to limit ourselves to crossover and mutation in an evolutionary algorithm. In principle, we can use any search operators to generate new solutions (i.e., offspring). A good search operator should always increase the probability of finding a global optimum. This is also true for selection.

4. Search Operators

There are many search operators that have been used in various evolutionary algorithms. Some of them are specialised in solving a particular

class of problems. This section describes some search operators and selection schemes[1] commonly used. They do not represent a complete set of all search operators.

4.1 Recombination Operators

The essence of any recombination (crossover) operator is the inheritance of information (genes) from two or more parents by offspring. Although most recombination operator uses two parents, multiple parents may be useful in some cases. Two offspring are often produced by a recombination operator, but, again, other numbers might be appropriate for some problems.

4.1.1 Recombination for Real-Valued Vectors. These operators are mostly proposed for evolution strategies. They are used to process vectors of real numbers. In evolution strategies, recombination is done independently for objective variables and strategy parameters (i.e., variance, etc.). It can be different for objective variables and strategy parameters.

Discrete Recombination In this case, an offspring vector will have components coming from two or more parent vectors. There is no change to any component itself. For example, given two parents $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ and $\mathbf{y} = (y_1, y_2, \ldots, y_n)$. The offspring $\mathbf{x}' = (x'_1, x'_2, \ldots, x'_n)$ and $\mathbf{y}' = (y'_1, y'_2, \ldots, y'_n)$ can be generated as follows:

$$x'_i = \begin{cases} x_i & \text{with probability } p_{recombination} \\ y_i & \text{otherwise} \end{cases}$$

$\mathbf{y}'$ will be the complement of $\mathbf{x}'$. A global version of this recombination is that y_i will be taken from a randomly generated $\mathbf{y}$ for each i value. That is, for each i value, a $\mathbf{y}$ is generated uniformly at random in the whole population. Then its ith component will be used for recombination. Basically the number of parents is the same as the population size.

Intermediate Recombination In this case, a component of an offspring vector is a linear combination (average) of parent's corresponding components. For example, given two parents $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ and $\mathbf{y} = (y_1, y_2, \ldots, y_n)$. The offspring $\mathbf{x}' = (x'_1, x'_2,$

[1]Usually selection is not regarded as a search operator. It is included in this section for ease of discussions.

$\ldots, x'_n)$ and $\mathbf{y}' = (y'_1, y'_2, \ldots, y'_n)$ can be generated as follows:

$$x'_i = x_i + \alpha(y_i - x_i)$$

where α is a weighting parameter in (0,1). It is traditionally set to 0.5. It can also be generated at random. $\mathbf{y}'$ can be generated similarly. A global version of this recombination is that y_i will be taken from a randomly generated $\mathbf{y}$ for each i value. α can also be different for each i.

4.1.2 Recombination for Binary Strings. Common recombination operators for binary strings include k-point crossover ($k > 1$) and uniform crossover, although there are many other variants.

k-point crossover This crossover can actually be applied to strings of any alphabet. Given two parents of length n. k random numbers, $r_1, r_2, \ldots, r_k$, between 1 and $n - 1$ will be generated uniformly (without repetition). Then an offspring is produced by taking segments (separated by $r_1, r_2, \ldots, r_k$) of parent strings alternately, i.e., the first segment from the first parent, the second from the second parent, the third from the first parent, and so on. For example, a 3-point crossover at 1,4,6 of two parents 00000000 and 11111111 will produce two offspring 01110011 and 10001100.

uniform crossover This crossover is also applicable to strings of any alphabet. An offspring is generated by taking its each bit or character from the corresponding bit or character in one of the two parents. The parent that the bit or character is to be taken from is chosen uniformly at random.

Other crossover operators include segmented crossover and shuffle crossover (Eshelman et al., 1989). They are not widely used in genetic algorithm applications.

4.1.3 Specialised Recombination. There are numerous recombination operators which have been proposed for different problems, especially combinatorial optimisation problems, such as matrix-based crossover (Vignaux and Michalewicz, 1991; Luan and Yao, 1994), permutation-based crossover (Whitley et al., 1991; Yao, 1993), tree-based crossover (Koza, 1992; Koza and Andre, 1998), etc.

4.2 Mutation Operators

Mutation operators used for vectors of real values are usually based on certain probability distributions, such as uniform, lognormal, Gauss

(normal) and Cauchy distributions. Mutation for binary strings is a lot simpler. It is usually a bit-fliping operation.

4.2.1 Mutation for Real-Valued Vectors.

Gaussian Mutation In this case, an offspring is produced by adding a Gaussian random number with mean 0 and standard deviation σ to the parent. For example, given $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ as the parent, an offspring is produced as follows:

$$x_i' = x_i + N_i(0, \sigma_i)$$

where $N_i(0, \sigma_i)$ is a normally distributed random number with mean 0 and standard deviation σ_i. The n random numbers are generated independently for each dimension (thus the subscription i in $N_i(0, \sigma_i)$. For self-adaptive evolutionary algorithms, such as evolution strategies and evolutionary programming, σ_i's are usually mutated independently using a lognormal distribution. More details are given by Eqs. 2.2 and 2.3 in Section 1.3.

Cauchy Mutation Cauchy mutation differs from Gaussian mutation in the probability distribution used to generate the random number. The use of Cauchy mutation in evolutionary algorithms was inspired by fast simulated annealing (Szu and Hartley, 1987; Yao, 1995) and proposed indepedently by several researchers (Yao and Liu, 1996; Kappler, 1996). A detailed study of Cauchy mutation will be presented later in this chapter.

Other Mutations Mutations based on other probability distributions, such as the *t*-distribution, may be introduced into evolutionary algorithms. An important question to ask when introducing a new operator is when the new operator will be most efficient for what kind of problems.

4.2.2 Mutation for Binary Strings.

Bit-Fliping Bit-flipping mutation simply flips a bit from 0 to 1 or from 1 to 0 with certain probability. This probability is often called the mutation probability or mutation rate. Bit-flipping mutation can be generalised to mutate strings of any alphabet. The generalised mutation works as follows: for each character (allele) in a string, replace it with another randomly chosen character (not the same as the one to be replaced) in the alphabet with certain mutation probability.

Random Bit This mutation does not flip a bit. It replaces a bit by 0 or 1 with equal probability (i.e., 0.5 respectively). The generalised version of this mutation works as follows: for each character (allele) in a string, replace it with a randomly chosen character (could be the same as the one to be replaced) in the alphabet with certain mutation probability.

Specialised Mutations Similar to the situation for crossover, there are many other specialised mutation operators designed for various combinatorial problems, such as the operators for mutate finite state machines (Fogel et al., 1966), artificial neural networks (Yao and Liu, 1997a) and cutting stock problems (Liang et al., 1998).

4.3 Selection

A selection scheme determines the probability of an individual being selected for producing offspring by recombination and/or mutation. In order to search for increasingly better individuals, fitter individuals should have higher probabilities of being selected while unfit individuals should be selected only with small probabilities. Different selection schemes have different methods of calculating selection probability. The selection pressure has sometimes been used to indicate how large the selection probability should be for a fit individual in comparison with that for an unfit individual. The larger the probability, the stronger the selection pressure.

There are three major types of selection schemes, roulette wheel selection (also known as the fitness proportional selection), rank-based selection and tournament selection.

4.3.1 Roulette Wheel Selection. Let $f_1, f_2, \ldots, f_n$ be fitness values of individuals $1, 2, \cdots, n$. Then the selection probability for individual i is

$$p_i = \frac{f_i}{\sum_{j=1}^{n} f_j}.$$

Roulette wheel selection calculates the selection probability directly from individual's fitness values.

This method may cause problems in some cases. For example, if an initial population contains one or two very fit but not the best individuals and the rest of the population are not good, then these fit individuals will quickly dominate the whole population (due to their very large selection probabilities) and prevent the population from exploring other potentially better individuals. On the other hand, if individuals in a population have very similar fitness values, it will be very difficult

for the population to move towards a better one since selection probabilities for fit and unfit individuals are very similar. To get around these two problems, various fitness scaling methods have been proposed (Goldberg, 1989). These fitness scaling methods are used to scale fitness values before they are used in calculating selection probabilities.

4.3.2 Rank-Based Selection. Rank-based selection does not calculate selection probabilities from fitness values directly. It sorts all individuals according to their fitness values first and then computes selection probabilities according to their ranks rather than their fitness values. Hence rank-based selection can maintain a constant selection pressure in the evolutionary search and avoid some of the problems encountered by roulette wheel selection.

There are many different rank-based selection schemes. Two are introduced here. Assume the best individual in a population ranks the first. The probability of selecting individual i can be calculated linearly as follows (Baker, 1985):

$$p_i = \frac{1}{n}\left(\eta_{max} - (\eta_{max} - \eta_{min})\frac{i-1}{n-1}\right)$$

where n is the population size, η_{max} and η_{min} are two parameters.

$$\eta_{max} \geq \eta_{min} \geq 0$$
$$\eta_{max} + \eta_{min} = 2$$

The recommended value for η_{max} is 1.1.

A rank-based selection scheme with a stronger selection pressure is the following nonlinear ranking scheme (Yao, 1993):

$$P_i = \frac{i}{\sum_{j=1}^{n} j}.$$

4.3.3 Tournament Selection. Both roulette wheel selection and rank-based selection are based on the global information in the whole population. This increases communications overheads if we want to parallelise an evolutionary algorithms on a parallel machine. Tournament selection only needs part of the whole population to calculate an individual's selection probability. Different individuals can also calculate their selection probabilities in parallel.

One of the often used tournament selection schemes is that used in evolutionary programming, which was described in Section 1.4. Another one is Boltzmann tournament selection (Goldberg, 1990), described as follows:

1 For tournament size 3, first select an individual i_1 at random. Then select i_2 also at random but must differ from i_1 by a fitness amount of Θ. Randomly selected i_3 must also differ from i_1, and half the time differ from i_2 as well, all by Θ.

2 i_2 competes with i_3 first. Then the winner competes with i_1. The winner is identified by the Boltzmann acceptance probability. The probability of individual x wining over y is:

$$P(x, y) = \frac{1}{1 + \exp((f_y - f_x)/T)}$$

where T is the temperature. (Note that we are maximising fitness.)

4.3.4 Elitist Selection. Elitist selection is also known as elitism and elitist strategy. It always copy the best individual to the next generation without any modification. More than one individual may be copied, i.e., the best, second best, etc., may be copied to the next generation without any modification. Elitism is usually used in addition to other selection schemes.

5. Summary

This chapter introduces the basic concept and major areas of evolutionary computation. It presents a brief history of three major types of evolutionary algorithms, i.e., evolution strategies, evolutionary programming and genetic algorithms, and points out similarities and differences among them. It is also pointed out that the field of evolutionary computation is much more than just three types of algorithms. The field includes many other topics.

The chapter gives a quick overview of evolutionary computation without diving into too much detail. Three main areas of the field have been discussed: evolutionary computation theory, evolutionary optimisation and evolutionary learning. It is argued that much work on the computational complexity of evolutionary algorithms is needed among other things in order to better understand the computational power of these algorithms.

References

Axelrod, R. (1987). The evolution of strategies in the iterated prisoner's dilemma. in *Genetic Algorithms and Simulated Annealing* (L. Davis, ed.), ch.-3, San Mateo, CA: Morgan Kaufmann, 32-41.

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

Bäck, T. and Schwefel, H.-P. (1993). An overview of evolutionary algorithms for parameter optimization. *Evolutionary Computation,* 1(1), 1-23.

Baker, J. E. (1985). Adaptive selection methods for genetic algorithms. in *Proc. of an Int'l Conf. on Genetic Algorithms and Their Applications* (J. J. Grefenstette, ed.), 101-111.

Beyer, H.-G. (1994). Towards a theory of evolution strategies: the (μ, λ) theory. *Evolutionary Computation,* 2(4), 381-407.

Colombetti, M. and Dorigo, M. (1998) Evolutionary computation in behavior engineering, in *Evolutionary Computation,* World Scientific Publ. Co.

Darwen, P. J. and Lao, X. (1995). On evolving robust strategies for iterated prisoner's dilemma, in *Progress in Evolutionary Computation, Lecture Notes in Artificial Intelligence, Vol. 956* (X. Lao, ed.), (Heidelberg, Germany), Springer-Verlag, 276-292.

Darwen, P. J. and Yao, X. (1996). Automatic modularization by speciation. in *Proc. of the 1996 IEEE Int'l Conf. on Evolutionary Computation (ICEC'96), Nagoya, Japan,* IEEE Press, New York, NY 10017-2394, 88-93.

Darwen, P. J. and Lao, X. (1997). Speciation as automatic categorical modularization. *IEEE Transactions on Evolutionary Computation,* 1(2), 101-108.

Davidor, Y. (1991). Epistasis variance: a viewpoint on GA-hardness. in *Foundations of Genetic Algorithms* (G. J. E. Rawlins, ed.), Morgan Kaufmann, San Mateo, CA, 23-35.

Deb, K. and Goldberg, D. E. (1993). Analyzing deception in trap functions, in *Foundations of Genetic Algorithms 2* (L. D. Whitley, ed.), (San Mateo, CA), Morgan Kaufmann, 93-108.

de Garis, H. (1990). Genetic programming: modular evolution for Darwin machines, in *Proc. of Int'l Joint Conf. on Neural Networks, Vol. I,* (Washington, DC), Lawrence Erlbaum Associates, Hillsdale, NJ, 194-197, .

DeJong, K. A. (1993). Genetic algorithms are NOT function optimizers. in *Foundations of Genetic Algorithms 2* (L. D. Whitley, ed.), (San Mateo, CA), Morgan Kaufmann, 5-17.

Eiben, A. E., Aarts, E. H. L. and van Hee, K. M. (1991). Global convergence of genetic algorithms: a markov chain analysis, in *Parallel Problem Solving from Nature* (H.-P. Schwefel and R. Manner, eds.), Springer-Verlag, Heidelberg, 4-12.

Eshelman, L., Caruana, R. and Schaffer, J. D. (1989). Biases in the crossover landscape. in *Proc. of the Third Int'l Conf. on Genetic Al-*

gorithms and Their Applications (J. D. Schaffer, ed.), Morgan Kaufmann, San Mateo, CA, 10-19.

Fogel, D. B. (1998). An evolutionary approach to the traveling salesman problem. *Biological Cybernetics,* 60, 139-144.

Fogel, D. B. (1991). *System Identification Through Simulated Evolution: A Machine Learning Approach to Modeling.* Needham Heights, MA 02194: Ginn Press.

Fogel, D. B. (1992). *Evolving Artificial Intelligence.* PhD thesis, University of California, San Diego, CA, 1992.

Fogel, D. B. (1994). An introduction to simulated evolutionary optimisation. *IEEE Trans, on Neural Networks,* 5(1), 3-14.

Fogel, D. B. (1995). *Evolutionary Computation: Towards a New Philosophy of Machine Intelligence.* New York, NY: IEEE Press.

Fogel, D. B. (1995a). A comparison of evolutionary programming and genetic algorithms on selected constrained optimization problems. *Simulation,* 64(6), 399-406.

Fogel, L. J., Owens, A. J. and Walsh, M. J. (1966). *Artificial Intelligence Through Simulated Evolution.* New York, NY: John Wiley & Sons.

Fogel D. B. and Atmar, J. W. (1990). Comparing genetic operators with Gaussian mutations in simulated evolutionary process using linear systems. *Biological Cybernetics,* 63(2), 111-114.

Fonseca, C. M. and Fleming, P. J. (1995). An overview of evolutionary algorithms in multiobjective optimization. *Evolutionary Computation,* 3(1), 1-16.

Fonseca, C. M. and Fleming, P. J. (1998). Multiobjective optimization and multiple constraint handling with evolutionary algorithms — part i: A unified formulation. *IEEE Trans. on Systems, Man, and Cybernetics, Part A: Systems and Humans,* 28(1), 26-37.

Fonseca, C. M. and Fleming, P. J. (1998a). Multiobjective optimization and multiple constraint handling with evolutionary algorithms — part ii: Application example. *IEEE Trans. on Systems, Man, and Cybernetics, Part A: Systems and Humans,* 28(1), 38-47.

Fraser, A. S. (1957). Simulation of genetic systems by automatic digital computers. I. Introduction. *Australian Journal of Biological Sciences,* 10, 484-491.

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

Glover, F. (1989). Tabu search — part I. *ORSA J. on Computing,* 1, 190-206.

Glover, F. (1990). Tabu search — part II. *ORSA J. on Computing,* 2, 4–32.

Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization, and Machine Learning.* Reading, MA: Addison-Wesley.

Goldberg, D. E. (1990). A note on Bltzmann tournament selection for genetic algorithms and population-oriented simulated annealing. *Complex Systems,* 4, 445-460.

Grefenstette, J. J. (1993). Deception considered harmful. in *Foundations of Genetic Algorithms 2* (L. D. Whitley, ed.), (San Mateo, CA), Morgan Kaufmann, 75-91.

Grefenstette, J. J., Gopal, R., Rosmaita, B. J. and van Gucht, D. (1985). Genetic algorithms for the traveling salesman problem. in *Proc. of the First Int'l Conf. on Genetic Algorithms and Their Applications* (J. J. Grefenstette, ed.), Carnegie-Mellon University, 160-168.

Heckendorn R. B. and Whitley, D. (1997). A Walsh analysis of NK-landscapes. in *Proc. of the 7th Int'l Conf. on Genetic Algorithms* (T. Bäck, ed.),(San Francisco, CA), Morgan Kaufmann, 41-48.

Hillis, W. D. (1991). Co-evolving parasites improve simulated evolution as an optimization procedure. in *Santa Fe Institute Studies in the Sciences of Complexity, Volume-10,* Addison-Wesley, 313-323.

Hinterding, R. and Khan, L. (1995). Genetic algorithms for cutting stock problems: with and without contiguity. in *Progress in Evolutionary Computation, Lecture Notes in Artificial Intelligence, Vol. 956* (X. Lao, ed.), (Heidelberg, Germany), Springer-Verlag, 166-186.

Holland, J. H. (1975). *Adaptation in Natural and Artificial Systems.* Ann Arbor, MI: The University of Michigan Press.

Holland, J. H., Holyoak, K. J., Nisbett, R. E. and Thagard, P. R. (1986). *Induction: Processes of Inference, Learning, and Discovery.* Cambridge, MA: The MIT Press.

Holland, J. H. (1988). Using classifier systems to study adaptive nonlinear networks. in *Lectures in the Sciences of Complexity* (D. L. Stein, ed.), Redwood City, CA: Addison-Wesley, 463-499.

Holland, J. H. (1992). *Adaptation in Natural and Artificial Systems (1st MIT Press Edn).* Cambridge, MA: The MIT Press.

Hordijk, W. (1996). A measure of landscapes. *Evolutionary Computation,*4(4), 335–360.

Ingber, L. (1989). Very fast simulated re-annealing. *Mathl. Comput. Modelling,* 12(8), 967-973.

Jones, T. (1995). *Evolutionary Algorithms, Fitness Landscapes and Search.* PhD thesis, The University of New Mexico, Albuquerque, New Mexico.

Jong, K. A. D. (1975). *An analysis of the behavior of a class of genetic adaptive systems.* PhD thesis, University of Michigan, Ann Arbor.

Kappler, C. (1996). Are evolutionary algorithms improved by large mutations?. in *Parallel Problem Solving from Nature (PPSN) IV* (H.-M. Voigt, W. Ebeling, I. Rechenberg, and H.-P. Schwefel, eds.), vol. 1141 of *Lecture Notes in Computer Science,* (Berlin), Springer-Verlag, 346-355.

Kido, T., Takagi, K. and Nakanishi, M. (1994). Analysis and comparisons of genetic algorithm, simulated annealing, tabu search, and evolutionary combination algorithm. *Informatica,* 18, 399-410.

Kim, J.-H. and Myung, H. (1997). Evolutionary programming techniques for constrained optimization problems. *IEEE Transactions on Evolutionary Computation,* 1(2), 129-140.

Kirkpatrick, S., Gelatt, C. D. and Vecchi, M. P. (1983). Optimization by simulated annealing. *Science,* 220, 671-680.

Koza, J. R. (1989). Evolving programs using symbolic expressions. in *Proc. of the 11th Int'l Joint Conf. on Artificial Intelligence* (N. S. Sridharan,ed.), (San Mateo, CA), Morgan Kaufmann, 768-774.

Koza, J. R. (1990). Genetic programming: a paradigm for genetically breeding populations of computer programs to solve problems. Tech. Rep. STAN-CS-90-1314, Department of Computer Science, Stanford University.

Koza, J. R. (1992). *Genetic Programming.* Cambridge, Mass.: The MIT Press.

Koza, J. and D. Andre, D. (1998). Automatic discovery of protein motifs using genetic programming, in *Evolutionary Computation,* World Scientific Publ. Co..

Liang, K.-H., Yao, X., Newton, C. and Hoffman, D. (1998). Solving cutting stock problems by evolutionary programming. in *Evolutionary Programming VII: Proc. of the 7th Annual Conference on Evolutionary Programming* (V. W. Porto, N. Saravanan, D. Waagen, and A. E. Eiben, eds.), vol. 1447 of *Lecture Notes in Computer Science,* (Berlin), Springer-Verlag, 291-300.

Lienig, J. and Thulasiraman, K. (1995). GASBOR: A genetic algorithm for switchbox routing in integrated circuits. in *Progress in Evolutionary Computation, Lecture Notes in Artificial Intelligence, Vol. 956* (X. Lao, ed.), (Heidelberg, Germany), Springer-Verlag, 187-200.

Liepins G. E. and Vose, M. D. (1991). Deceptiveness and genetic algorithm dynamics. in *Foundations of Genetic Algorithms* (G. J. E. Rawlins, ed.), 1, (San Mateo, CA), Morgan Kaufmann, 36-50.

Liu, Y. and Lao, X. (1996). A population-based learning algorithm which learns both architectures and weights of neural networks. *Chinese Journal of Advanced Software Research (Allerton Press, Inc., New York, NY 10011),* 3(1), 54-65.

Luan, F. and Yao, X. (1994). Solving real-world lecture room assignment problems by genetic algorithms. in *Complex Systems — From Local Interactions to Global Phenomena,* IOS Press, Amsterdam, 148-160.

Manderick, B., de Weger, M. and Spiessens, P. (1991). The genetic algorithm and the structure of the fitness landscape. in *Proc. of the Fourth Int'l Conf. on Genetic Algorithms* (R. K. Belew and L. B. Booker, eds.), Morgan Kaufmann, San Mateo, CA, 143-150.

Mathias K. E. and Whitley, D. (1994). Changing representations during search: A comparative study of delta coding. *Evolutionary Computation,* 2(3), 249-278.

Michalewicz, Z. (1992). *Genetic Algorithms + Data Structures = Evolution Programs.* Berlin, Germany: Springer-Verlag.

Michalewicz, Z. (1996). *Genetic Algorithms + Data Structures = Evolution Programs (3rd edition).* Berlin, Germany: Springer-Verlag.

Michalewicz, Z. and Schoenauer, M. (1996). Evolutionary algorithms for constrained parameter optimization problems. *Evolutionary Computation,* 4(1), 1-32.

Reynolds, D. and Gomatam, J. (1996). Stochastic modelling of genetic algorithms. *Artificial Intelligence,* 82(1-2), 303-330.

Rosin, C. D. and Belew, R. K. (1997). New methods for competitive coevolution. *Evolutionary Computation,* 5(1), 1-29.

Rudolph, G. (1994) Convergence properties of canonical genetic algorithms. *IEEE Trans. on Neural Networks,* 5(1), 96-101.

Rudolph, G. (1994a). Convergence of non-elitist strategies. in *Proc. of the 1994 IEEE Int'l Conf. on Evolutionary Computation (ICEC'94)* (Z. M. et al.,ed.), IEEE Press, New York, NY 10017-2394, 63-66.

Rudolph, G. (1996). Convergence analysis of evolutionary algorithms in general search spaces, in *Proc. of the 1996 IEEE Int'l Conf. on Evolutionary Computation (ICEC'96),* (New York, NY), IEEE Press, 50-54.

Schwefel, H.-P. (1981). *Numerical Optimization of Computer Models.* Chichester: John Wiley & Sons.

Schwefel, H.-P. (1995). *Evolution and Optimum Seeking.* New York: John Wiley & Sons.

Smith, R. E. and Goldberg, D. E. (1992). Reinforcement learning with classifier systems: adaptive default hierarchy formation. *Applied Artificial Intelligence,* 6, 79-102.

Suzuki, J. (1993). A markov chain analysis on a genetic algorithm. in *Proc. of the Fifth Int'l Conf. on Genetic Algorithms* (S. Forrest, ed.), Morgan Kaufmann, San Mateo, CA, 146-153.

Suzuki, J. (1995). A markov chain analysis on a simple genetic algorithm. *IEEE Trans. on Systems, Man, and Cybernetics,* 25, 655-659.

Szu, H. H. and Hartley, R. L. (1987). Fast simulated annealing. *Physics Letters A,* 122, 157-162.

Vignaux, G. A. and Michalewicz, Z. (1991). A genetic algorithm for the linear transportation problem. *IEEE Trans, on Systems, Man, and Cybernetics,* 21(2), 445-452.

Westerdale, T. H. (1997). Classifier systems — no wonder they don't work. in *Proc. of the 2nd Annual Conf. on Genetic Programming* (J. R. Koza, K. Deb, M. Dorigo, D. B. Fogel, M. Garzon, H. Iba, and R. L. Riolo, eds.), (San Francisco, CA), Morgan Kaufmann, 529-537.

Whitley, D., Starkweather, T. and Shaner, D. (1991). The traveling salesman and sequence scheduling: quality solutions using genetic edge recombination. in *Handbook of Genetic Algorithms* (L. Davis, ed.), ch. 22, New York, NY: Van Nostrand Reinhold, 350-372.

Wilson, S. W. (1995). Classifier fitness based on accuracy. *Evolutionary Computation,* 3(2), 149-175.

Yao, X. (1991). Optimization by genetic annealing, in *Proc. of Second Australian Conf. on Neural Networks* (M. Jabri, ed.), (Sydney, Australia), 94-97.

Yao, X. (1991a). Simulated annealing with extended neighbourhood. *Int. J. of Computer Math.,* 40, 169-189.

Yao, X. (1991b). Evolution of connectionist networks, in *Preprints of the Int'l Symp. on AI, Reasoning & Creativity* (T. Dartnall, ed.), (Queensland, Australia), Griffith University, 49-52.

Yao, X. (1993). An empirical study of genetic operators in genetic algorithms. *Microprocessing and Microprogramming,* 38, 707-714.

Yao, X. (1993a). A review of evolutionary artificial neural networks. *International Journal of Intelligent Systems,* 8(4), 539-567.

Yao, X. (1995). A new simulated annealing algorithm. *Int. J. of Computer Math.,* 56, 161-168.

Yao, X. (1995a). Evolutionary artificial neural networks. in *Encyclopedia of Computer Science and Technology* (A. Kent and J. G. Williams, eds.), vol.-33, New York, NY 10016: Marcel Dekker Inc., 137-170.

Yao, X. and Darwen, P. (1994). An experimental study of N-person iterated prisoner's dilemma games. *Informatica,* 18, 435-450.

Yao, X. and Liu, Y. (1996). Fast evolutionary programming. in *Evolutionary Programming V: Proc. of the Fifth Annual Conference on Evolutionary Programming* (L. J. Fogel, P. J. Angeline, and T. Back, eds.), (Cambridge, MA), The MIT Press, 451-460.

Yao, X. and Liu, Y. (1997). Fast evolution strategies. *Control and Cybernetics,* 26(3), 467-496.

Yao, X. and Liu, Y. (1997a). A new evolutionary system for evolving artificial neural networks. *IEEE Transactions on Neural Networks,* 8(3), 694-713.

Yao, X. and Liu, Y. (1998). Making use of population information in evolutionary artificial neural networks. *IEEE Trans. on Systems, Man, and Cybernetics, Part B: Cybernetics,* 28(3), 417-425.

Yao, X. and Shi, Y. (1995). A preliminary study on designing artificial neural networks using co-evolution. in *Proc. of the IEEE Singapore Intl Conf on Intelligent Control and Instrumentation,* (Singapore), IEEE Singapore Section, 149-154.

This page intentionally left blank

II

SINGLE OBJECTIVE OPTIMIZATION

This page intentionally left blank

Chapter 3

EVOLUTIONARY ALGORITHMS AND CONSTRAINED OPTIMIZATION

Zbigniew Michalewicz and
Martin Schmidt

Abstract Evolutionary computation techniques have received a lot of attention regarding their potential as optimization techniques for complex numerical functions. However, they have not produced a significant breakthrough in the area of nonlinear programming due to the fact that they have not addressed the issue of constraints in a systematic way. Only during the last decade several methods have been proposed for handling nonlinear constraints by evolutionary algorithms for numerical optimization problems; however, these methods give different performance on different test cases.

In this chapter we (1) present some issues which should be addressed while solving the general nonlinear programming problem, (2) survey several approaches which have emerged in the evolutionary computation community, and (3) discuss briefly a methodology, which may serve as a handy reference for future methods.

1. Introduction

Every real-world problem poses constraints. You can't get away from them. It's only the textbooks that allow you solve problems in the absence of constraints. Dhar and Ranganathan (1990) wrote:

> Virtually all decision making situations involve constraints. What distinguishes various types of problems is the form of these constraints. Depending on how the problem is visualized, they can arise as rules, data dependencies, algebraic expressions, or other forms.

We would only amend this by removing the word "virtually!" As we can face problems with very pesky constraints, in this chapter we concentrate on a variety of constraint-handling techniques that might be incorporated into evolutionary algorithms. In this case we have the po-

tential of treating both feasible and infeasible solutions simultaneously within a single evolving population.

In evolutionary computation methods the evaluation function serves as the main link between the problem and the algorithm by rating individual solutions in the population. Superior individuals are usually given higher probabilities for survival and reproduction. It's crucial that we define the evaluation function to capture and characterize the problem in a reasonable way. This can be a significant challenge when facing the possibility of having infeasible solutions. Our final answer must be a *feasible* solution, otherwise it's really no *solution* at all. It might be useful, however, to operate on infeasible solutions while searching for better feasible solutions. Finding a proper evaluation measure for feasible and infeasible individuals is of paramount importance. It can mean the difference between success or failure.

Usually constraints cause some difficulties in most problem-solving strategies, but it needn't always be the case. Sometimes constraints are helpful and can guide you in the right direction. We'll explore some possibilities for taking advantage of constraints later in the chapter. First, let's tackle some of the general issues that are connected with handling constraints in evolutionary algorithms. Later we illustrate many of these issues in the domain of nonlinear programming problems (NLPs). Finally, we discuss briefly a new tool for evaluating all existing constraint-handling methods: a test case generator.

2. General considerations

When facing constrained optimization problems using evolutionary algorithms, it's very important to process infeasible individuals (Michalewicz, 1995). Particularly in real-world problems, you'll find it difficult to design operators that avoid them entirely while still being effective in locating useful feasible solutions. In general, a search space $\mathcal{S}$ consists of two disjoint subsets of feasible and infeasible subspaces, $\mathcal{F}$ and $\mathcal{U}$, respectively (see figure 3.1). Here, we're not making any assumptions about these subspaces. In particular, they don't have to be convex or even connected (e.g., as shown in figure 3.1 where the feasible part $\mathcal{F}$ of the search space consists of four disjoint subsets). In solving optimization problems we search for a *feasible* optimum. During the search process we have to deal with various feasible and infeasible individuals. For example (see figure 3.1), at some stage of the evolution, a population might contain some feasible individuals (b, c, d, e, j) and infeasible individuals $(a, f, g, h, i, k, l, m, n, o)$, while the (global) optimum solution is marked by "X".

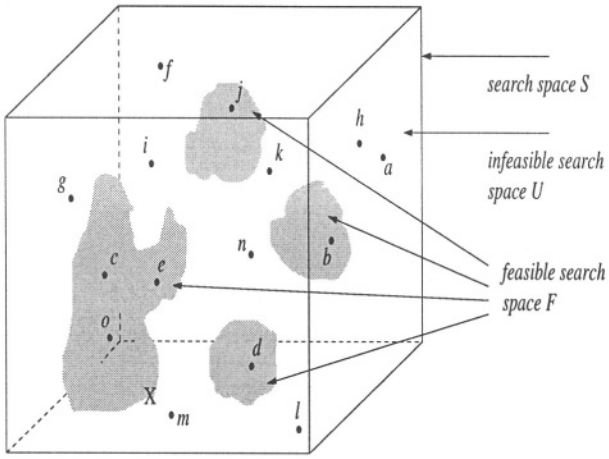


Figure 3.1. A search space and its feasible and infeasible parts with a population of 15 individuals, $a - o$

Having to operate on both feasible and infeasible solutions can affect how we design various aspects of an evolutionary algorithm. Suppose we were using some form of elitist selection. Should we simply maintain the best *feasible* individual, or should we perhaps maintain an *infeasible* individual that happens to score better by our evaluation function? Questions sometimes arise in designing variation operators as well. Some operators might only be applicable to feasible individuals. But without a doubt, the major concern is the design of a suitable evaluation function to treat both feasible and infeasible solutions. This is far from trivial.

In general, we'll have to design two evaluation functions, $eval_f$ and $eval_u$, for the feasible and infeasible domains, respectively. There are many important questions to be addressed:

1. How should we compare two feasible individuals, e.g., "c" and "j" from figure 3.1? In other words, how should we design the function $eval_f$?

2. How should we compare two infeasible individuals, e.g., "a" and "n"? In other words, how should we design the function $eval_u$?

3. How are the functions $eval_f$ and $eval_u$ related? Should we assume, for example, that $eval_f(s) \succ eval_u(r)$ for any $s \in \mathcal{F}$ and any $r \in \mathcal{U}$? (The symbol $\succ$ is interpreted as "is better than," i.e., "greater than" for maximization and "less than" for minimization problems.)

4. Should we consider infeasible individuals harmful and eliminate them from the population?

5. Should we "repair" infeasible solutions by moving them into the closest point of the feasible space (e.g., the repaired version of "*m*" might be the optimum "X," figure 3.1)?

6. If we repair infeasible individuals, should we replace an infeasible individual by its repaired version in the population or should we instead only use a repair procedure for evaluations?

7. Since our aim is to find a feasible optimum solution, should we choose to penalize infeasible individuals?

8. Should we start with an initial population of feasible individuals and maintain the feasibility of their offspring by using specialized operators?

9. Should we change the topology of the search space by using decoders that might translate infeasible solutions into feasible solutions?

10. Should we extract a set of constraints that define the feasible search space and process individuals and constraints separately?

11. Should we concentrate on searching a boundary between feasible and infeasible parts of the search space?

12. How should we go about finding a feasible solution?

Several trends for handling infeasible solutions have emerged in evolutionary computation; most of these have only come about quite recently, making efforts from a decade or more ago almost obsolete (e.g., Richardson et al., 1989). Even when using penalty functions to degrade the quality of infeasible solutions, this area of application now consists of several methods that differ in many important respects. Other newer methods maintain the feasibility of the individuals in the population by means of specialized operators or decoders, impose a restriction that any feasible solution is "better" than any infeasible solution, consider constraints one at the time in a particular order, repair infeasible solutions, use inxmultiobjective optimization techniques, are based on cultural algorithms, or rate solutions using a particular co-evolutionary model. We'll discuss briefly these techniques by addressing issues 1 – 12 in turn.

2.1 Feasible solutions

For textbook problems, designing the evaluation function f is usually easy: it's usually given to you. For example, when treating most operations research problems, such as inxknapsack problems, the inxTSP, inxset covering, and so forth, the evaluation function comes part-and-parcel along with the problem. But when dealing with the real world, things aren't always so obvious. For example, in many design problems

there are no clear formulas for comparing two feasible designs. Some problem-dependent heuristics are necessary in these cases, which should provide a numerical measure $eval_f(x)$ of the quality of a feasible individual x.

Even in classic combinatorial optimization problems such as bin packing, we can encounter some difficulties. As noted in Falkenauer (1994), where the task was to pack some maximum number of potentially different items into bins of various size, the obvious evaluation function of counting the number of bins that suffices to pack all of the items is insufficient. The reason it's insufficient is that the resulting "landscape" to be searched isn't very friendly. There is a relatively small number of optimum solutions and a very large number of solutions that evaluate to just one higher number (i.e., they require just one more bin). All of these suboptimal solutions have the same perceived quality, so how can we traverse the space when there's no guidance from the evaluation function on which directions to go? Clearly, the problem of designing a "perfect" $eval_f$ is far from trivial.

Actually, there's an entirely different possibility, because in many cases we don't have to define the evaluation function $eval_f$ to be a mapping to the real numbers. In a sense, we don't have to define it at all! It's really only mandatory if we're using a selection method that acts on the solutions' values, such as proportional selection. For other types of selection, however, all that might be required is an ordering relation that says that one solution is better than another. If an ordering relation $\succ$ handles decisions of the type "is a feasible individual x better than a feasible individual y?" then such a relation $\succ$ is sufficient for tournament and ranking selection methods, which require either selecting the best individual out of some number of individuals, or a rank ordering of all individuals, respectively.

2.2 Infeasible solutions

Designing the evaluation function for treating infeasible individuals is quite difficult. It's tempting to avoid it altogether by simply rejecting infeasible solutions (see section 2.4). Alternatively, we can extend the domain of the function $eval_f$ in order to handle infeasible individuals, i.e., $eval_u(x) = eval_f(x) \pm Q(x)$, where $Q(x)$ represents either a penalty for the infeasible individual x, or a cost for repairing such an individual (i.e., converting it to a feasible solution, see section 2.7). Another option is to design a separate evaluation function $eval_u$ that's independent of $eval_f$; however, we then have to establish some relationship between these two functions (see section 2.3).

Evaluating infeasible individuals presents quite a challenge. Consider a knapsack problem, where your goal is to get as many items as possible into a knapsack of some particular size. The amount by which you violate the capacity of the knapsack might not be a very good measure of that particular solution's "fitness." This also holds true for many scheduling, timetabling, and planning problems.

As with feasible solutions, it's possible to develop an ordering relation for infeasible individuals (as opposed to constructing $eval_u$). In both cases it's necessary to establish a relationship between the evaluations of feasible and infeasible individuals.

2.3 Feasible vs. infeasible solutions

Let's say that we've decided to treat both feasible and infeasible individuals and evaluate them using two evaluation functions, $eval_f$ and $eval_u$, respectively. That is, a feasible individual x is evaluated using $eval_f(x)$ and an infeasible individual y is evaluated using $eval_u(y)$. Now we have to establish a relationship between these two evaluation functions.

As mentioned previously, one possibility is to design $eval_u$ based on $eval_f$, $eval_u(y) = eval_f(y) \pm Q(y)$, where $Q(y)$ represents either a penalty for the infeasible individual y, or a cost for repairing such an individual (see section 2.7). Alternatively, we could construct a global evaluation function $eval$ as

$$eval(p) = \begin{cases} q_1 \cdot eval_f(p) & if \ \ p \in \mathcal{F} \\ q_2 \cdot eval_u(p) & if \ \ p \in \mathcal{U}. \end{cases}$$

In other words, we have two weights, q_1 and q_2, that are used to scale the relative importance of $eval_f$ and $eval_u$.

Note that both of the methods provide the possibility for an infeasible solution to be "better" than a feasible solution. That is, it's possible to have a feasible individual x and an infeasible individual y such that $eval(y) \succ eval(x)$. This may lead the algorithm to converge to an infeasible solution. This phenomenon has led many researchers to experiment with dynamic penalty functions Q (see section 2.7) that increase the pressure on infeasible solutions as the evolutionary search proceeds. The problem of selecting $Q(x)$ (or weights q_1 and q_2), however, can be as difficult as solving the original problem itself.

2.4 Rejecting infeasible solutions

The "death penalty" heuristic is a popular option in many evolutionary algorithms. Simply kill off all infeasible solutions at every step. Note

that rejecting infeasible individuals does simplify things. For example, there's no need to design $eval_u$ and to compare it with $eval_f$.

Eliminating infeasible solutions may work well when the feasible search space is convex and constitutes a reasonable part of the whole search space. Otherwise this approach has serious limitations. For example, there are many search problems where a random sampling of solutions may generate an initial population that's entirely infeasible. It would therefore be essential to improve these solutions instead of reject them outright. Wiping the slate clean doesn't help here because you're right back where you started. Moreover, for many variation operators, it's quite often the case that the evolutionary algorithm can reach the optimum solution more quickly if it can "cross" an infeasible region (this is especially true in nonconvex feasible search spaces).

2.5 Repairing infeasible solutions

The idea of repairing infeasible solutions enjoys a particular popularity in the evolutionary computation community, and especially so for certain combinatorial optimization problems (e.g., TSP, knapsack problem, set covering problem). In these cases, it's relatively easy to repair an infeasible solution and make it feasible. Such a repaired version can be used either for evaluation, i.e., $eval_u(y) = eval_f(x)$, where x is a repaired (i.e., feasible) version of y, or it can replace the original individual in the population (perhaps with some probability, see section 2.6). Note that the repaired version of solution "m" (figure 3.1) might be the optimum "X".

The process of repairing infeasible individuals is related to a combination of learning and evolution (the so-called *Baldwin effect* (Whitely et al., 1996)). Learning (as local search in general, and local search for the closest feasible solution, in particular) and evolution interact with each other. The fitness value of the local improvement is transferred to the individual. In that way a local search is analogous to the learning that occurs during a single generation.

The weakness of these methods lies in their problem dependence. Different repair procedures have to be designed for each particular problem. There are no standard heuristics for accomplishing this. It's sometimes possible to use a greedy method for repairing solutions, and at other times you might use some random procedure, or a variety of alternative choices. And then sometimes repairing infeasible solutions might become as complex a task as solving the original problem. This is often the case in nonlinear transportation problems, scheduling, and timetabling.

2.6 Replacement of solutions

The idea of replacing repaired individuals is related to what's called *Lamarckian evolution* (Whitely et al., 1996), which assumes that an individual improves during its lifetime and that the resulting improvements are coded back into the genetics of that individual. This is, of course, not the way nature works, but remember that we're designing evolutionary algorithms as computational procedures for solving problems, and we don't have to be constrained by the way nature works.

Orvosh and Davis (1993) reported a so-called 5-percent-rule, which states that in many combinatorial optimization problems, when coupling a repair procedure with an evolutionary algorithm, the best results are achieved when 5 percent of repaired individuals replace their infeasible original versions. We know that this rule can't work in all problem domains, but this is at least a starting point to try when facing some new problem. In continuous domains, a new replacement rule emerged: The GENOCOP III system (Michalewicz and Nazhiyath, 1995) that uses a repair function appears to work well on certain problems when repairing 15 percent of the individuals. Higher and lower percentages have yielded inferior performance. Again, these types of settings are problem dependent and might even vary while a problem is being solved.

2.7 Penalizing infeasible solutions

The most common approach to handling infeasible solutions is to provide a penalty for their infeasibility by extending the domain of $eval_f$, and assuming that $eval_u(p) = eval_f(p) \pm Q(p)$, where $Q(p)$ represents either a penalty for an infeasible individual p, or a cost for repairing such an individual. The primary question then concerns how we should design such a penalty function $Q(p)$. Intuition suggests that the penalty should be kept as low as possible, just above the threshold below which infeasible solutions are optimal (the so-called *minimal penalty rule* (Leriche et al., 1995), But it's often difficult to implement this rule effectively.

The relationship between an infeasible individual "p" and the feasible part $\mathcal{F}$ of the search space $\mathcal{S}$ plays a significant role in penalizing such individuals. An individual might be penalized just for being infeasible, for the "amount" of its infeasibility, or for the effort required to repair the individual. For example, for a knapsack problem with a weight capacity of 99 kilograms we may have two infeasible solutions that yield the same profit (which is calculated based on the items you can fit in the knapsack), where the total weight of all items taken is 100 and 105 kilograms, respectively. It's difficult to argue that the first individual with the total weight of 100 is better than the other one with a total

weight of 105, despite the fact that for this individual the violation of the capacity constraint is much smaller than for the other one. The reason is that the first solution may involve five items each weighing 20 kilograms, and the second solution may contain (among other items) an item of low profit and a weight of six kilograms: Removing this item would yield a feasible solution, one that might be much better than any repaired version of the first individual. In these cases, the appropriate penalty function should consider the ease of repairing an individual, as well as the quality of the repaired version. Again, this is problem dependent.

2.8 Maintaining feasibility

It seems that one of the most reasonable heuristics for dealing with the issue of feasibility is to use specialized representations and variation operators to maintain the feasibility of individuals in the population.

Several specialized systems have been developed for particular optimization problems. These evolutionary algorithms rely on unique representations and specialized variation operators. Some examples were described in (Davis, 1991) and many others are described here. For example, GENOCOP (Michalewicz and Janikow, 1996) assumes that the problem you face has only linear constraints and a feasible starting point (or a feasible initial population). A closed set of variation operators maintains the feasibility of solutions. There's no need to ever treat infeasible solutions when these conditions hold.

Very often such systems are much more reliable than other evolutionary techniques based on a penalty approach. Many practitioners have used problem-specific representations and specialized variation operators in numerical optimization, machine learning, optimal control, cognitive modeling, classical operation research problems (TSP, knapsack problems, transportation problems, assignment problems, bin packing, scheduling, partitioning, etc.), engineering design, system integration, iterated games, robotics, signal processing, and many others. The variation operators are often tailored to the representation (e.g., Fogel et al., 1966, Koza, 1992).

2.9 Using decoders

Using some form of decoder offers an interesting option when designing an evolutionary algorithm. In these methods, the data structure that represents an individual doesn't encode for a solution directly, but instead provides the instruction for how to build a feasible solution.

It's important to point out several factors that should be taken into account while using decoders. Each decoder imposes a relationship T between a feasible solution and a decoded solution. It's important that several conditions are satisfied: (1) for each solution $s \in \mathcal{F}$ there is an encoded solution d, (2) each encoded solution d corresponds to a feasible solution s, and (3) all solutions in $\mathcal{F}$ should be represented by the same number of encodings d.[1] Additionally, it's reasonable to request that (4) the transformation T is computationally fast and (5) it has a locality feature in the sense that small changes in the encoded solution result in small changes in the solution itself. An interesting study on coding trees in evolutionary algorithms was reported by Palmer and Kershenbaum (1994), where the above conditions were formulated.

2.10 Separating solutions and constraints

Separating out individuals and constraints is a general and interesting heuristic. The first possibility includes the use of multiobjective optimization methods, where the evaluation function f and constraint violation measures f_j (for m constraints) constitute an $(m+1)$-dimensional vector $\mathbf{v} = (f, f_1, \ldots, f_m)$. Using some multiobjective optimization method, we can attempt to minimize its components. An ideal solution x would have $f_j(x) = 0$ for $1 \leq i \leq m$ and $f(x) \leq f(y)$ for all feasible y (minimization problems). Surry et al. (1995) presented a successful implementation of this approach.

Another heuristic is based on the idea of handling constraints in a particular order. Schoenauer and Xanthakis (1993) called this method a "behavioral memory" approach.

2.11 Exploring boundaries

A special approach was proposed in evolutionary algorithms where variation operators can be designed to search the boundary (and only the boundary) between feasible and infeasible solutions (Michalewicz et al., 1996). The general concept for boundary operators is that all individuals of the initial population lie on the boundary between feasible and

[1] As observed by Davis (1995), however, the requirement that all solutions in $\mathcal{F}$ should be represented by the same number of decodings seems overly strong. There are cases in which this requirement might be suboptimal. For example, suppose we have a decoding and encoding procedure that makes it impossible to represent suboptimal solutions, and which encodes the optimal one. This might be a good thing. An example would be a graph coloring order-based structure with a decoding procedure that gives each node its first legal color. This representation couldn't encode solutions where some nodes that could be colored in fact weren't colored, but this is a good thing!

infeasible areas of the search space and the operators are closed with respect to the boundary. Then all the offspring are at the boundary points as well. As these operators were proposed in the context of constrained parameter optimization, we discuss this approach in the next section.

2.12 Finding feasible solutions

There are problems for which any feasible solution would be of value. These are tough problems! Here, we really aren't concerned with any optimization issues (finding *the best* feasible solution) but instead we have to find *any* point in the feasible search space $\mathcal{F}$. These problems are called *constraint satisfaction problems.*

Some evolutionary algorithms have been designed to tackle these problems. Paredis experimented with two different approaches to constraint satisfaction problems. The first approach (Paredis, 1992 and 1993) was based on a clever representation of individuals where each component was allowed to take on values from the search domain, as well as an additional value of '?,' which represented choices that remained undecided. The initial population consisted of strings that possessed all ?s. A selection-assignment-propagation cycle then replaced some ? symbols by values from the appropriate domain (the assignment is checked for consistency with the component in question). The quality of such partially-defined individuals was defined as the value of the evaluation function of the best complete solution found when starting the search from a given partial individual. Variation operators were extended to incorporate a repair process (a constraint-checking step). This system was implemented and executed on several N-queens problems (Paredis, 1993) as well as some scheduling problems (Paredis, 1992).

In the second approach, Paredis (1994) investigated a co-evolutionary model, where a population of potential solutions co-evolves with a population of constraints. Fitter solutions satisfy more constraints, whereas fitter constraints are violated by more solutions. This means that individuals from the population of solutions are considered from the whole search space, and that there's no distinction between feasible and infeasible individuals. The evaluation of an individual is determined on the basis of constraint violation measures f_js; however, better f_js (e.g., active constraints) contribute more towards the evaluation of solutions. The approach was tested on the N-queens problem and compared with other single-population approaches (Paredis, 1994 and 1995).

3. Numerical optimization

Let us concentrate now on one particular area of applications of evolutionary algorithms: numerical optimization. There have been many efforts to use evolutionary algorithms for constrained numerical optimization problems (Michalewicz, 1994). In this section we survey some approaches of addressing constraints; these approaches would illustrate some points raised in the previous section of this chapter and would provide more concrete examples.

In general, constraint-handling techniques which have been incorporated in evolutionary algorithms can be grouped into five basic categories:

1 Methods based on preserving the feasibility of solutions.

2 Methods based on penalty functions.

3 Methods that make a clear distinction between feasible and infeasible solutions.

4 Methods based on decoders.

5 Other hybrid methods.

We've just explored many of the issues involved in each of these categories. Now let's revisit and take a look at methods in more detail. Furthermore, for most of the methods enumerated here, we'll provide a test case and the result of the method on that case.

3.1 Preserving the feasibility

There are two methods that fall in this category. Let's discuss them in turn.

Use of specialized operators. The idea behind the GENOCOP system[2] (Michalewicz, 1996; Michalewicz and Janikow, 1996) is based on specialized variation operators that transform feasible individuals into other feasible individuals. These operators are closed on the feasible part $\mathcal{F}$ of the search space. As noted earlier, the method assumes that we are facing only linear constraints and that we have a feasible starting point (or population). Linear equations are used to eliminate some variables, which are replaced as a linear combination of the remaining

[2]GENOCOP stands for GEnetic algorithm for Numerical Optimization of COnstrained Problems and was developed before the different branches of evolutionary algorithms became functionally similar.

variables. Linear inequalities are updated accordingly. For example, when a particular component x_i of a solution vector $\mathbf{x}$ is varied, the evolutionary algorithm determines its current domain $dom(x_i)$ (which is a function of linear constraints and the remaining values of the solution vector $\mathbf{x}$) and the new value of x_i is taken from this domain (either with a uniform probability distribution or other probability distributions for nonuniform and boundary-type variations). Regardless of the specific distribution, it's chosen such that the offspring solution vector is always feasible. Similarly, arithmetic crossover, $a\mathbf{x} + (1 - a)\mathbf{y}$, on two feasible solution vectors $\mathbf{x}$ and $\mathbf{y}$ always yields a feasible solution (for $0 \leq a \leq 1$) in convex search spaces. Since this evolutionary system assumes only linear constraints, this implies the convexity of the feasible search space $\mathcal{F}$.

Searching the boundary of the feasible region. Searching along the boundaries of the feasible-infeasible regions can be very important when facing optimization problems with nonlinear equality constraints or with active nonlinear constraints at the target optimum. Within evolutionary algorithms, there's a significant potential for incorporating specialized operators that can search such boundaries efficiently. We provide one example of such an approach; for more details see (Schoenauer and Michalewicz, 1996).

Consider the following numerical optimization problem: maximize the function

$$G(\mathbf{x}) = \left|\frac{\sum_{i=1}^{n} \cos^4(x_i) - 2\prod_{i=1}^{n} \cos^2(x_i)}{\sqrt{\sum_{i=1}^{n} i x_i^2}}\right|,$$

subject to

$\prod_{i=1}^{n} x_i \geq 0.75$, $\sum_{i=1}^{n} x_i \leq 7.5n$, and bounds $0 \leq x_i \leq 10$ for $1 \leq i \leq n$.

Function G is nonlinear and its global maximum is unknown, lying somewhere near the origin. The problem has one nonlinear constraint and one linear constraint. The latter is inactive around the origin and will be omitted. So the boundary between feasible and infeasible regions is defined by the equation $\prod x_i = 0.75$. The problem is difficult and no standard methods gave satisfactory results. This function was the first for which the idea of searching only the boundary was tested (Michalewicz et al., 1996). Specific initialization procedures and variation operators could be tailored to the problem owing to the simple analytical formulation of the constraint.

- **Initialization.** Randomly choose a positive variable for x_i, and use its inverse as a variable for x_{i+1}. The last variable is either 0.75 (when n is odd), or is multiplied by 0.75 (if n is even), so that the point lies on the boundary surface.

- **Crossover.** The variation operator of *geometrical crossover* is defined by $(x_i)(y_i) \rightarrow (x_i^{\alpha} y_i^{1-\alpha})$, with α randomly chosen in [0,1].

- **Mutation.** Pick two variables randomly, multiply one by a random factor $q > 0$ and the other by $\frac{1}{q}$ (restrict q to respect the bounds on the variables). ΛMutation

3.2 Penalty functions

The main efforts to treat constrained evolutionary optimization have involved the use of (extrinsic) penalty functions that degrade the quality of an infeasible solution. In this manner, the constrained problem is made unconstrained by using the modified evaluation function

$$eval(\mathbf{x}) = \begin{cases} f(\mathbf{x}), & \text{if } \mathbf{x} \in \mathcal{F} \\ f(\mathbf{x}) + penalty(\mathbf{x}), & \text{otherwise,} \end{cases}$$

where $penalty(\mathbf{x})$ is zero if no violation occurs, and is positive, otherwise (assuming the goal is minimization). The *penalty* function is usually based on some form of distance that measures how far the infeasible solution is from the feasible region $\mathcal{F}$, or on the effort to "repair" the solution, i.e., to transform it into $\mathcal{F}$. Many methods rely on a set of functions f_j $(1 \leq j \leq m)$ to construct the penalty, where the function f_j measures the violation of the j-th constraint:

$$f_j(\mathbf{x}) = \begin{cases} \max\{0, g_j(\mathbf{x})\}, & \text{if } 1 \leq j \leq q \\ |h_j(\mathbf{x})|, & \text{if } q+1 \leq j \leq m. \end{cases}$$

But these methods differ in many important details with respect to how the penalty function is designed and applied to infeasible solutions. Some more details on the specifics is provided below.

Method of static penalties. Homaifar et al. (1994) proposed that a family of intervals be constructed for every constraint that we face, where the intervals determine the appropriate penalty coefficient. The idea works as follows:

(1) For each constraint, create several (ℓ) levels of violation.
(2) For each level of violation and for each constraint, create a penalty coefficient R_{ij} $(i = 1, 2, \ldots, \ell,\ j = 1, 2, \ldots, m)$. Higher levels of

violation require larger values of this coefficient (again, we assume minimization).
(3) Start with a random population of individuals (feasible or infeasible).
(4) Evolve the population. Evaluate individuals using

$$eval(\mathbf{x}) = f(\mathbf{x}) + \sum_{j=1}^{m} R_{ij} f_j^2(\mathbf{x}).$$

The weakness of the method arises in the number of parameters. For m constraints the method requires $m(2\ell + 1)$ parameters in total: m parameters to establish the number of intervals for each constraint, ℓ parameters for each constraint thereby defning the boundaries of the intervals (the levels of violation), and ℓ parameters for each constraint representing the penaltycoefficients R_{ij}. In particular, for $m = 5$ constraints and $\ell = 4$ levels of violation, we need to set 45 parameters! And the results are certainly parameter dependent. For any particular problem, there might exist an optimal set of parameters for which we'd generate feasible near-optimum solutions, but this set might be very difficult to find.

Method of dynamic penalties. In contrast to Homaifar et al. (1994), Joines and Houck (1994) applied a method that used dynamic penalties. Individuals are evaluated at each iteration (generation) *t,* by the formula

$$eval(\mathbf{x}) = f(\mathbf{x}) + (C \times t)^{\alpha} \sum_{j=1}^{m} f_j^{\beta}(\mathbf{x}),$$

where C, α, and β are constants. A reasonable choice for these parameters is $C = 0.5$, $\alpha = \beta = 2$ (Joines. and Houck, 1994). This method doesn't require as many parameters as the static method described earlier, and instead of defining several violation levels, the selection pressure on infeasible solutions increases due to the $(C \times t)^{\alpha}$ component of the penalty term: as t grows larger, this component also grows larger.

Method of annealing penalties. A different means for dynamically adjusting penalties can take a clue from the annealing algorithm: Perhaps we can anneal the penalty values using a parameter that is analogous to "temperature." This procedure was incorporated into the second version of GENOCOP (GENOCOP II) (Michalewicz. and Attia, 1994):
(1) Divide all the constraints into four subsets: linear equations, linear inequalities, nonlinear equations, and nonlinear inequalities.
(2) Select a random single point as a starting point. (The initial population consists of copies of this single individual.) This initial point satisfies the linear constraints.
(3) Set the initial temperature $\tau = \tau_0$.
(4) Evolve the population using $eval(\mathbf{x}, \tau) = f(\mathbf{x}) + \frac{1}{2\tau} \sum_{j=1}^{m} f_j^2(\mathbf{x})$,

(5) If $\tau < \tau_f$, stop; otherwise,
– Decrease the temperature τ.
– Use the best available solution to serve as a starting point for the next iteration.
– Repeat the previous step of the algorithm.

This method distinguishes between linear and nonlinear constraints. The algorithm maintains the feasibility of all linear constraints using a set of closed operators that convert a feasible solution (feasible, that is, only in terms of linear constraints) into another feasible solution. The algorithm only considers active constraints at every generation and the selective pressure on infeasible solutions increases due to the decreasing values of the temperature τ.

Methods of adaptive penalties. Instead of relying on a fixed decreasing temperature schedule, it's possible to incorporate feedback from the search into a means for adjusting the penalties. Let's discuss two possibilities.

Bean and Hadj-Alouane (Bean and Hadj-Alouane, 1992, Hadj-Alouane and Bean, 1992) developed a method where the penalty function takes feedback from the search. Each individual is evaluated by the formula

$$eval(\mathbf{x}) = f(\mathbf{x}) + \lambda(t) \sum_{j=1}^{m} f_j^2(\mathbf{x}),$$

where $\lambda(t)$ is updated at every generation t:

$$\lambda(t+1) = \begin{cases} (1/\beta_1) \cdot \lambda(t), & \text{if } \mathbf{b}^i \in \mathcal{F} \text{ for all } t-k+1 \le i \le t \\ \beta_2 \cdot \lambda(t), & \text{if } \mathbf{b}^i \in \mathcal{S}-\mathcal{F} \text{ for all } t-k+1 \le i \le t \\ \lambda(t), & \text{otherwise,} \end{cases}$$

where $\mathbf{b}^i$ denotes the best individual in terms of function *eval* at generation i, $\beta_1, \beta_2 > 1$ and $\beta_1 \neq \beta_2$ to avoid cycling. In other words, (1) if all of the best individuals in the last k generations were feasible the method decreases the penalty component $\lambda(t+1)$ for generation $t+1$, and (2) if all of the best individuals in the last k generations were infeasible then the method increases the penalties. If there are some feasible and infeasible individuals as best individuals in the last k generations, $\lambda(t+1)$ remains without change.

A different method was offered by Smith and Tate (1993) that uses a "near-feasible" threshold q_j for each constraint $1 \le j \le m$. These thresholds indicate the distances from the feasible region $\mathcal{F}$ that are considered "reasonable" (or, in other words, that qualify as "interesting" infeasible solutions because they are close to the feasible region). Thus

the evaluation function is defined as

$$eval(\mathbf{x}, t) = f(\mathbf{x}) + F_{feas}(t) - F_{all}(t) \sum_{j=1}^{m} (f_j(\mathbf{x})/q_j(t))^k,$$

where $F_{all}(t)$ denotes the unpenalized value of the best solution found so far (up to generation t), $F_{feas}(t)$ denotes the value of the best feasible solution found so far, and k is a constant. Note, that the near-feasible thresholds $q_j(t)$ are dynamic. They are adjusted during the search based on feedback from the search. For example, it's possible to define $q_j(t) = q_j(0)/(1 + \beta_j t)$ thus increasing the penalty component over time. To the best of our knowledge, neither of the adaptive methods described in this subsection has been applied to continuous nonlinear programming problems.

Death penalty method. The death penalty method simply rejects infeasible solutions, killing them immediately (see above). This simple method can provide quality results for some problems. This method requires initialization with feasible solutions so comparisons to other methods can be tricky, but an interesting pattern emerged from the experiments analyzed in Michalewicz (1995). Simply rejecting infeasible methods performed quite poorly and wasn't as robust as other techniques (i.e., the standard deviation of the solution values was relatively high).

Segregated evolutionary algorithms. When tuning penalty coefficients, too small a penalty level leads to solutions that are infeasible because some penalized solutions will still exhibit an apparently higher quality than some feasible solutions. On the other hand, too high a penalty level restricts the search to the feasible region and thereby foregoes any short cut across the infeasible region. This often leads to premature stagnation at viable solutions of lesser value. One method for overcoming this concern was offered in Leriche et al. (1995).

The idea is to design two different penalized evaluation functions with static penalty terms p_1 and p_2. Penalty p_1 is purposely too small, while penalty p_2 is hopefully too high. Every individual in the current population undergoes recombination and mutation (or some form of variation). The values of the two evaluation functions $f_i(\mathbf{x}) = f(\mathbf{x}) + p_i(\mathbf{x})$, $i = 1, 2$, are computed for each resulting offspring (at no extra cost in terms of evaluation function calls), and two ranked lists are created according to the evaluated worth of all parents and offspring for each evaluation function.

3.3 Search for feasible solutions

Some evolutionary methods emphasize a distinction between feasible and infeasible solutions. One method considers the problem constraints in sequence. Once a sufficient number of feasible solutions is found in the presence of one constraint, the next constraint is considered. Another method assumes that any feasible solution is better than any infeasible solution (as we discussed above). Yet another method repairs infeasible individuals. Let's take each of these examples in turn.

Behavioral memory method. Schoenauer and Xanthakis (1993) proposed what they call a "behavioral memory" approach:
(1) Start with a random population of individuals (feasible or infeasible).
(2) Set $j = 1$ (j is a constraint counter).
(3) Evolve this population with $eval(\mathbf{x}) = f_j(\mathbf{x})$ until a given percentage of the population (a so-called flip threshold ϕ) is feasible for this constraint.[3]
(4) Set $j = j + 1$.
(5) The current population is the starting point for the next phase of the evolution, where $eval(\mathbf{x}) = f_j(\mathbf{x})$ (defined in the section 3.2). During this phase, points that don't satisfy one of the first, second, ..., or $(j-1)$-th constraints are eliminated from the population. The halting criterion is again the satisfaction of the j-th constraint using the flip threshold percentage ϕ of the population.
(6) If $j < m$, repeat the last two steps, otherwise ($j = m$) optimize the evaluation function, i.e., $eval(\mathbf{x}) = f(\mathbf{x})$, while rejecting infeasible individuals.

The method requires a sequential ordering of all constraints that are processed in turn. The influence of the order of constraints on the end results isn't very clear, and different orderings can generate different results both in the sense of total running time and accuracy. In all, the method requires three parameters: the sharing factor σ, the flip threshold ϕ, and a particular order of constraints. The method's very different from many others, and really is quite different from other penalty approaches since it only considers one constraint at a time. Also, the method concludes by optimizing using the "death penalty" approach, so it can't be neatly parceled into one or another category.

[3]The method suggests using a so-called sharing scheme to maintain population diversity.

Method of superiority of feasible points. Powell and Skolnick (1993) used a method that's based on a classic penalty approach, but with one notable exception. Each individual is evaluated by the formula

$$eval(\mathbf{x}) = f(\mathbf{x}) + r\sum_{j=1}^{m} f_j(\mathbf{x}) + \theta(t, \mathbf{x}),$$

where r is a constant; however, the original component $\theta(t, \mathbf{x})$ is an additional iteration-dependent function that influences the evaluations of infeasible solutions. The point is that the method distinguishes between feasible and infeasible individuals by adopting an additional heuristic (suggested earlier Richardson et al., 1989). For any feasible individual $\mathbf{x}$ and any infeasible individual $\mathbf{y}$: $eval(\mathbf{x}) < eval(\mathbf{y})$, i.e., any feasible solution is better than any infeasible one. This can be achieved in many ways. One possibility is to set

$$\theta(t, \mathbf{x}) = \begin{cases} 0, & \text{if } \mathbf{x} \in \mathcal{F} \\ \max\{0, \max_{\mathbf{x}\in\mathcal{F}}\{f(\mathbf{x})\} \\ \qquad - \min_{\mathbf{x}\in\mathcal{S}-\mathcal{F}}\{f(\mathbf{x}) + r\sum_{j=1}^{m} f_j(\mathbf{x})\}\}, & \text{otherwise.} \end{cases}$$

Infeasible individuals are penalized such that they can't be better than the worst feasible individual (i.e., $\max_{\mathbf{x}\in\mathcal{F}}\{f(\mathbf{x})\}$).[4]

In a recent study (Deb, 1999), this approach was modified using tournament selection coupled with the evaluation function

$$eval(\mathbf{x}) = \begin{cases} f(\mathbf{x}), & \text{if } \mathbf{x} \text{ is feasible,} \\ f_{\max} + \sum_{j=1}^{m} f_j(\mathbf{x}), & \text{otherwise,} \end{cases}$$

where $f_{\max}$ is the function value of the worst feasible solution in the population. The main difference between this approach and Powell and Skolnick's approach is that here the evaluation function value is not considered in evaluating an infeasible solution. Additionally, a niching scheme might be introduced to maintain diversity among feasible solutions. Thus, the search focuses initially on finding feasible solutions and then, when an adequate sampling of feasible solutions have been found, the algorithm finds better feasible solutions by maintaining a diverse set of solutions in the feasible region. There's no need for penalty coefficients here because the feasible solutions are always evaluated to be better than infeasible solutions, and infeasible solutions are compared purely on the basis of their constraint violations. Normalizing the constraints $f_j(\mathbf{x})$ is suggested.

Repairing infeasible individuals. GENOCOP III (the successor to the previous GENOCOP systems), incorporates the ability to repair infeasible solutions, as well as some of the concepts of co-evolution

[4]Powell and Skolnick (1993) achieved the same result by mapping evaluations of feasible solutions into the interval $(-\infty, 1)$ and infeasible solutions into the interval $(1, \infty)$. This difference in implementation isn't important for ranking and tournament selection methods.

(Michalewicz and Nazhiyath, 1995). As with the original GENOCOP, linear equations are eliminated, the number of variables is reduced, and linear inequalities are modified accordingly. All points included in the initial population satisfy linear constraints. Specialized operators maintain their feasibility in the sense of linear constraints from one generation to the next. We denote the set of points that satisfy the linear constraints by $\mathcal{F}_l \subseteq \mathcal{S}$.

Nonlinear equations require an additional parameter (γ) to define the precision of the system. All nonlinear equations $h_j(\mathbf{x}) = 0$ (for $j = q+1, \ldots, m$) are replaced by a pair of inequalities $-\gamma \leq h_j(\mathbf{x}) \leq \gamma$. Thus, we only deal with nonlinear inequalities. These nonlinear inequalities further restrict the set $\mathcal{F}_l$. They define the fully feasible part $\mathcal{F} \subseteq \mathcal{F}_l$ of the search space $\mathcal{S}$.

GENOCOP III extends GENOCOP by maintaining two separate populations where a development in one population influences the evaluations of individuals in the other. The first population P_s consists of so-called search points from $\mathcal{F}_l$ that satisfy the linear constraints of the problem. As mentioned earlier, the feasibility of these points, in the sense of linear constraints, is maintained by specialized operators.

The second population P_r consists of so-called reference points from $\mathcal{F}$; these points are fully feasible, i.e., they satisfy *all* the constraints.[5] Reference points $\mathbf{r}$ from P_r, being feasible, are evaluated directly by the evaluation function, i.e., $eval(\mathbf{r}) = f(\mathbf{r})$. On the other hand, search points from P_s are "repaired" for evaluation and the repair process works as follows. Assume there's a search point $\mathbf{s} \in P_s$. If $\mathbf{s} \in \mathcal{F}$, then $eval(\mathbf{s}) = f(\mathbf{s})$, since $\mathbf{s}$ is fully feasible. Otherwise (i.e., $\mathbf{s} \notin \mathcal{F}$), the system selects[6] one of the reference points, say $\mathbf{r}$ from P_r, and creates a sequence of random points $\mathbf{z}$ from a segment between $\mathbf{s}$ and $\mathbf{r}$ by generating random numbers a from the range $\langle 0, 1\rangle$: $\mathbf{z} = a\mathbf{s} + (1-a)\mathbf{r}$.[7] Once a fully feasible $\mathbf{z}$ is found, $eval(\mathbf{s}) = eval(\mathbf{z}) = f(\mathbf{z})$.[8]

Additionally, if $f(\mathbf{z})$ is better than $f(\mathbf{r})$, then the point $\mathbf{z}$ replaces $\mathbf{r}$ as a new reference point in the population of reference points P_r. Also, $\mathbf{z}$ replaces $\mathbf{s}$ in the population of search points P_s with some probability of replacement p_r.

---

[5] If GENOCOP III has difficulties in locating such a reference point for the purpose of initialization, it prompts the user for it. In cases where the ratio $|\mathcal{F}|/|\mathcal{S}|$ of feasible points in the search space is very small, it may happen that the initial set of reference points consists of multiple copies of a single feasible point.

[6] Better reference points have greater chances of being selected. A nonlinear ranking selection method was used.

[7] Note that all such generated points $\mathbf{z}$ belong to $\mathcal{F}_l$.

[8] The same search point $\mathcal{S}$ can evaluate to different values in different generations due to the random nature of the repair process.

3.4 Decoders

Decoders offer an interesting alternative for designing evolutionary algorithms, but they've only been applied in continuous domains recently (Koziel and Michalewicz, 1998 and 1999). It's relatively easy to establish a one-to-one mapping between an arbitrarily convex feasible search space $\mathcal{F}$ and the n-dimensional cube $[-1, 1]^n$ (see figure 3.2).

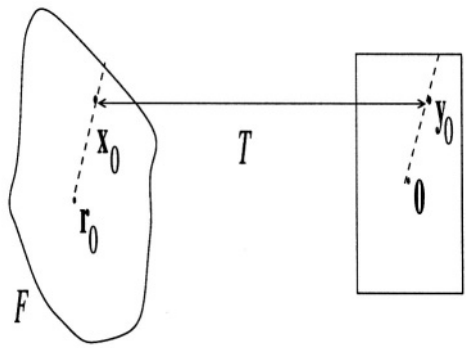


Figure 3.2. A mapping T from a space $\mathcal{F}$ into a cube $[-1, 1]^n$ (two-dimensional case)

Note that an arbitrary point (other than **0**) $\mathbf{y}_0 = (y_{0,1}, \ldots, y_{0,n}) \in [-1, 1]^n$ defines a line segment from the **0** to the boundary of the cube. This segment is described by $y_i = y_{0,i} \cdot t$, for $i = 1, \ldots, n$, where t varies from 0 to $t_{max} = 1/\max\{|y_{0,1}|, \ldots, |y_{0,n}|\}$. For $t = 0$, $\mathbf{y} = \mathbf{0}$, and for $t = t_{max}$, $\mathbf{y} = (y_{0,1}t_{max}, \ldots, y_{0,n}t_{max})$ — a boundary point of the $[-1, 1]^n$ cube. Consequently, the corresponding feasible point (to $\mathbf{y}_0 \in [-1, 1]^n$) $\mathbf{x}_0 \in \mathcal{F}$ (with respect to some reference point[9] $\mathbf{r}_0$) is defined as $\mathbf{x}_0 = \mathbf{r}_0 + \mathbf{y}_0 \cdot \tau$, where $\tau = \tau_{max}/t_{max}$, and τ_{max} is determined with arbitrary precision by a binary search procedure such that $\mathbf{r}_0 + \mathbf{y}_0 \cdot \tau_{max}$ is a boundary point of the feasible search space $\mathcal{F}$.

The above mapping satisfies all the requirements of a "good" decoder. Apart from being one-to-one, the transformation is fast and has a "locality" feature (i.e., points that are close before being mapped are close after being mapped).

[9]A reference point $\mathbf{r}_0$ is an arbitrary internal point of the convex set $\mathcal{F}$. Note, that it's not necessary for the feasible search space $\mathcal{F}$ to be convex. Assuming the existence of the reference point $\mathbf{r}_0$, such that every line segment originating in $\mathbf{r}_0$ intersects the boundary of $\mathcal{F}$ in precisely one point, is sufficient. This requirement is satisfied, of course, for any convex set $\mathcal{F}$.

3.5 Hybrid methods

It's easy to develop hybrid methods that combine evolutionary algorithms with deterministic procedures for numerical optimization problems. For example, Waagen et al. (1992) combined a floating-point representation and Gaussian variation with the direction set method of Hooke-Jeeves. This hybrid was tested on three unconstrained continuous-valued test functions. Myung et al. (1995) considered a similar approach but experimented with constrained problems in the continuous domain. Again, they used floating-point representations and Gaussian variation, but combined this with a Lagrange method developed by Maa and Shanblatt (1992) into a two-phased search. During the first phase, the evolutionary algorithm optimizes the function

$$eval(\mathbf{x}) = f(\mathbf{x}) + \frac{s}{2}\left(\sum_{j=1}^{m} f_j^2(\mathbf{x})\right),$$

where s is a constant. After this phase terminates, the second phase applies the Maa and Shanblatt procedure to the best solution found during the first phase. This second phase iterates until the system

$$\mathbf{x}' = -\nabla f(\mathbf{x}) - \left[\sum_{j=1}^{m} \nabla f_j(\mathbf{x})(s f_j(\mathbf{x}) + \lambda_j)\right],$$

is in equilibrium, where the Lagrange multipliers are updated as $\lambda_j' = \epsilon s f_j$ for a small positive constant ϵ.

Several other constraint-handling methods also deserve attention. For example, some methods use the values of the evaluation function f and penalties f_j $(j = 1, \ldots, m)$ as elements of a vector and apply multiobjective techniques to minimize all the components of the vector. For example, Schaffer (1984) selects $1/(m+1)$ of the population based on each of the objectives. Such an approach was incorporated by Parmee and Purchase (1994) in the development of techniques for constrained design spaces. On the other hand, Surry et al. (1995), ranked all the individuals in the population on the basis of their constraint violations. This rank, r, together with the value of the evaluation function f, leads to the two-objective optimization problem. This approach gave a good performance on optimization of gas supply networks (Surry et al., 1995).

Hinterding and Michalewicz (1998) used a vector of constraint violations, to assist in the process of parent selection, where the length of the vector corresponded to the number of constraints. For example, if the first parent satisfies constraints 1, 2, and 4 (say out of 5), it's mated preferably with an individual that satisfies constraints 3 and 5. It's also possible to incorporate knowledge of the problem constraints into the belief space of cultural algorithms (Reynolds, 1994). These algorithms provide a possibility for conducting an efficient search of the feasible search space (Reynolds et al., 1995).

4. Final Remarks

We've covered a great many facets of constrained optimization problems and provided a survey of some of the attempts to treat these problems. Even though it might seem that this survey was lengthy, in fact, it was overly brief. It only skimmed the surface of what has been done with evolutionary algorithms and what remains to be done. Nevertheless, we have to admit that the characteristics that make a constrained problem difficult for an evolutionary algorithm (or for that matter, other methods) aren't clear. Problems can be characterized by various parameters: the number of linear constraints, the number of nonlinear constraints, the number of equality constraints, the number of active constraints, the ratio $\rho = |\mathcal{F}|/|\mathcal{S}|$ of the size of feasible search space to the whole, and the type of evaluation function in terms of the number of variables, the number of local optima, the existence of derivatives, and so forth.

In Michalewicz and Schoenauer (1996), 11 test cases ($G1$–$G11$) for constrained numerical optimization problems were proposed. These test cases include evaluation functions of various types (linear, quadratic, cubic, polynomial, nonlinear) with various numbers of variables and different types (linear inequalities, nonlinear equalities and inequalities) and numbers of constraints. The ratio ρ between the size of the feasible search space $\mathcal{F}$ and the size of the whole search space $\mathcal{S}$ for these test cases varies from zero to almost 100 percent. The topologies of feasible search spaces are also quite different.

Even though many constraint-handling methods reported successes on particular test cases, the results of many tests haven't provided meaningful patterns to predict the difficulty of problems. No single parameter, such as the number of linear, nonlinear, active constraints, and so forth, can suffice to describe the problem difficulty. Several methods were also quite sensitive to the presence of a feasible solution in the initial population. There's no doubt that more extensive testing and analysis is required. The question of how to make an appropriate choice of an evolutionary method for a nonlinear optimization problem *a priori* remains open. It seems that more complex properties of the problem (e.g., the characteristic of the evaluation function together with the topology of the feasible region) may constitute quite significant measures of the difficulty of the problem. Also, some additional measures of the problem characteristics due to the constraints might be helpful. So far, we don't have this sort of information at hand.

Michalewicz and Schoenauer (1996) offered:

> It seems that the most promising approach at this stage of research is experimental, involving the design of a scalable test suite of constrained optimization problems, in which many [...] features could be easily

tuned. Then it should be possible to test new methods with respect to the corpus of all available methods.

There's a clear need for a parameterized test-case generator that can be used for analyzing various methods in a systematic way instead of testing them on a few selected cases. Furthermore, it's not clear if the addition of a few extra specific test cases is really of any help. A few test-case generators have been just constructed.

There have been some attempts to propose a test case generator for unconstrained parameter optimization (Whitley et al., 1995, 1996). We are also aware of some attempts to do so for constrained cases. In Van Kemenade (1998) the author proposed so-called stepping-stones problem defined as:

$$\text{maximize } \sum_{i=1}^{n}(x_i/\pi + 1),$$

where $-\pi \leq x_i \leq \pi$ for $i = 1, \ldots, n$ and the following constraints are satisfied:

$$e^{x_i/\pi} + cos(2x_i) \leq 1 \text{ for } i = 1, \ldots, n.$$

Note that the evaluation function is linear and that the feasible region is split into 2^n disjoint parts (called stepping-stones). As the number of dimensions n grows, the problem becomes more complex. However, as the stepping-stones problem has only one parameter, it can not be used to investigate some aspects of a constraint-handling method.

In Michalewicz (1999) a test-case generator for constrained parameter optimization techniques was proposed. This generator is capable of creating various test cases with different characteristics:

- problems with different value of ρ: the relative size of the feasible region in the search space
- problems with different number and types of constraints
- problems with convex or non-convex evaluation function, possibly with multiple optima
- problems with highly non-convex constraints consisting of (possibly) disjoint regions.

All this can be achieved by setting a few parameters that influence different characteristics of the optimization problem. Such a test-case generator should be very useful for analyzing and comparing different constraint-handling techniques.

However, this test case generator is far from perfect. It defines a landscape which is a collection of site-wise optimizable functions, each defined on different subspaces of equal sizes. Because of that all basins of attractions have the same size, moreover, all points at the boundary

between two basins of attraction have the same value. The local optima are located in centers of the hypercubes; all feasible regions are centered around the local optima. Note also, that while we can change the number of constraints, there is precisely one active constraint at the global optimum.

In Schmidt and Michalewicz (2000) a new version of the test case generator was defined to overcome the limitations of the previous one. Here it is possible to control dimensionality, multimodality, ruggedness of the landscape, the number of feasible components, and the size of the feasible search space. With the gradual and intuitive control over its parameters, the proposed test case generator is a significant improvement over earlier versions. It should be very useful in evaluating (in a systematic way) various constraints handling methods; it should allow to understand their merits and drawbacks.

References

Bean, J. C. and A. B. Hadj-Alouane (1992). A dual genetic algorithm for bounded integer programs. Technical Report TR 92-53, Department of Industrial and Operations Engineering, The University of Michigan.

Bowen, J. and G. Dozier (1995). Solving Constraint Satisfaction Problems Using a Genetic/Systematic Search Hybrid that Realizes When to Quit. In *Proceedings of the Sixth International Conference on Genetic Algorithms,* Eshelman, L.J.(ed.), Morgan Kaufmann, San Mateo, CA., 122-129.

Davis, L., ed. (1991). *Handbook of Genetic Algorithms.* Van Nostrand Reinhold, NY.

Davis, L. (1995). Private communication.

Deb, K. (1999). An Efficient Constraint Handling Method for Genetic Algorithms. *Computer Methods in Applied Mechanics and Engineering,* in press.

Dhar, V. and Ranganathan, N., (1990). *Integer Programming vs. Expert Systems: An Experimental Comparison,* Communications of the ACM, 33(3), 323-336.

Eiben, A.E., P.-E. Raue, and Zs. Ruttkay (1994). Solving Constraint Satisfaction Problems Using Genetic Algorithms. In *Proceedings of the 1994 IEEE Conference on Evolutionary Computation,* IEEE Press, Piscataway, NJ, 542-547.

Falkenauer, E. (1994). A New Representation and Operators for GAs Applied to Grouping Problems. *Evolutionary Computation,* 2(2), 123-144.

Fogel, L.J., A.J. Owens, and M.J. Walsh (1966). *Artificial Intelligence Through Simulated Evolution.* John Wiley, New York, NY.

Hadj-Alouane, A. B. and J. C. Bean (1992). A genetic algorithm for the multiple-choice integer program. Technical Report TR 92-50, Department of Industrial and Operations Engineering, The University of Michigan.

Hinterding, R. and Z. Michalewicz (1998). Your Brains and My Beauty: Parent Matching for Constrained Optimisation. In *Proceedings of the 1998 IEEE Conference on Evolutionary Computation,* IEEE Press, Piscataway, NJ, 810-815.

Homaifar, A., S. H.-Y. Lai, and X. Qi (1994). Constrained optimization via genetic algorithms. *Simulation 62*(4), 242–254.

Joines, J. and C. Houck (1994). On the use of non–stationary penalty functions to solve nonlinear constrained optimization problems with gas. In Z. Michalewicz, J. D. Schaffer, H.-P. Schwefel, D. B. Fogel, and H. Kitano (Eds.), *Proceedings of the First IEEE International Conference on Evolutionary Computation,* IEEE Press, 579–584.

Keane, A.J. (1996). A Brief Comparison of Some Evolutionary Optimization Methods. In *Modern Heuristic Search Methods,* V. Rayward-Smith, I. Osman, C. Reeves and G. D. Smith, eds., John Wiley, New York, NY, 255-272.

van Kemenade, C.H.M. (1998). Recombinative evolutionary search. PhD Thesis, Leiden University, Netherlands.

Koza, J.R. (1992). *Genetic Programming.* MIT Press, Cambridge, MA.

Koziel, S. and Z. Michalewicz (1998). A Decoder-Based Evolutionary Algorithm for Constrained Parameter Optimization Problems. In *Proceedings of the 5th Parallel Problem Solving from Nature Conference,* Eiben, A.E., T. Bäck, M. Schoenauer, and H.–P. Schwefel, (eds.), Lecture Notes in Computer Science, Vol.1498, Springer, Berlin, 231-240.

Koziel, S. and Z. Michalewicz (1999). Evolutionary Algorithms, Homomorphous Mappings, and Constrained Parameter Optimization. *Evolutionary Computation,* 7(1), 19-44.

Leriche, R. G., C. Knopf-Lenoir, and R. T. Haftka (1995). A segragated genetic algorithm for constrained structural optimization. In L. J. Eshelman (Ed.), *Proceedings of the* 6^{th} *International Conference on Genetic Algorithms,* 558-565.

Maa, C. and M. Shanblatt (1992). A two-phase optimization neural network. *IEEE Transactions on Neural Networks,* 3(6), 1003-1009.

Michalewicz, Z. (1995a). Genetic algorithms, numerical optimization and constraints. In L. J. Eshelman (Ed.), *Proceedings of the* 6^{th} *International Conference on Genetic Algorithms,* Morgan Kaufmann, 151-158.

Michalewicz, Z. (1993). A Hierarchy of Evolution Programs: An Experimental Study. *Evolutionary Computation,* 1(1), 51-76.

Michalewicz, Z. (1994). Evolutionary Computation Techniques for Nonlinear Programming Problems. *International Transactions in Operational Research,* 1(2), 223-240.

Michalewicz, Z. (1995). Heuristic Methods for Evolutionary Computation Techniques. *Journal of Heuristics,* 1(2), 177-206.

Michalewicz, Z. (1996). *Genetic Algorithms+Data Structures=Evolution Programs.* New-York: Springer Verlag. 3rd edition.

Michalewicz, Z. and N. Attia (1994). Evolutionary optimization of constrained problems. In *Proceedings of the 3rd Annual Conference on Evolutionary Programming,* World Scientific, 98-108.

Michalewicz, Z. (1995). Genetic Algorithms, Numerical Optimization and Constraints. In *Proceedings of the Sixth International Conference on Genetic Algorithms,* Eshelman, L.J.(ed.), Morgan Kaufmann, San Mateo, CA., 151-158.

Michalewicz, Z., K. Deb, M. Schmidt, and T. Stidsen (2000). Test Case Generator for Constrained Parameter Optimization Techniques. IEEE Transactions on Evolutionary Computation.

Michalewicz, Z. and C. Z. Janikow (1991). Handling constraints in genetic algorithms. In R. K. Belew and L. B. Booker (Eds.), *Proceedings of the 4th International Conference on Genetic Algorithms,* Morgan Kaufmann, 151-157.

Michalewicz, Z., T. Logan, and S. Swaminathan (1994). Evolutionary operators for continuous convex parameter spaces. In *Proceedings of the 3rd Annual Conference on Evolutionary Programming,* World Scientific, 84-97.

Michalewicz, Z. and Michalewicz, M. (1995). Pro-Life versus Pro-Choice Strategies in Evolutionary Computation Techniques. Chapter 10 in *Evolutionary Computation,* IEEE Press.

Michalewicz, Z. and G. Nazhiyath (1995). GENOCOP III: A Coevolutionary Algorithm for Numerical Optimization Problems with Nonlinear Constraints. In *Proceedings of the 1995 IEEE Conference on Evolutionary Computation,* IEEE Press, Piscataway, NJ, 647-651.

Michalewicz, Z., G. Nazhiyath, and M. Michalewicz (1996). A Note on Usefulness of Geometrical Crossover for Numerical Optimization Problems. In *Proceedings of the 5th Annual Conference on Evolutionary Programming,* Fogel, L.J., P.J. Angeline, and T. Back, (eds.), MIT Press, Cambridge, MA, 305-312.

Michalewicz, Z. and M. Schoenauer (1996). Evolutionary Algorithms for Constrained Parameter Optimization Problems. *Evolutionary Computation,* 4(1), 1-32.

Michalewicz, Z. and C. Janikow, C. (1996). GENOCOP: A Genetic Algorithm for Numerical Optimization Problems with Linear Constraints. *Communications of the ACM,* December, 118.

Myung, H., J.-H. Kim, and D.B. Fogel (1995). Preliminary Investigation Into a Two-stage Method of Evolutionary Optimization on Constrained Problems. In *Proceedings of the 4th Annual Conference on Evolutionary Programming,* McDonnell, J.R., R.G. Reynolds, and D.B. Fogel, (eds.), MIT Press, Cambridge, MA, 449-463.

Orvosh, D. and L. Davis (1993). Shall we repair? Genetic algorithms, combinatorial optimization, and feasibility constraints. In S. Forrest (Ed.), *Proceedings of the* 5^{th} *International Conference on Genetic Algorithms,* Morgan Kaufmann, 650.

Palmer, C.C. and A. Kershenbaum (1994). Representing Trees in Genetic Algorithms. In *Proceedings of the IEEE International Conference on Evolutionary Computation,* 27-29 June 1994, 379-384.

Paredis, J. (1992). Exploiting Constraints as Background Knowledge for Genetic Algorithms: A Case-Study for Scheduling. In *Proceedings of the 2nd Conference on Parallel Problem Solving from Nature 2.,* Männer, R. and B. Manderick, (eds.), North-Holland, Amsterdam, The Netherlands , 229-238.

Paredis, J. (1993). Genetic State-Space Search for Constrained Optimization Problems. In *Proceedings of the 13th International Joint Conference on Artificial Intelligence,* Morgan Kaufmann, San Mateo, CA.

Paredis, J. (1994). Coevolutionary constraint satisfaction. In Y. Davidor, H.-P. Schwefel, and R. Manner (Eds.), *Proceedings of the* 3^{rd} *Conference on Parallel Problems Solving from Nature,* Springer Verlag, 46-55.

Paredis, J. (1995). The Symbiotic Evolution of Solutions and Their Representations. In *Proceedings of the Sixth International Conference on Genetic Algorithms,* Eshelman, L.J.(ed.), Morgan Kaufmann, San Mateo, CA., 359-365.

Parmee, I. and G. Purchase (1994). The development of directed genetic search technique for heavily constrained design spaces. In *Proceedings of the Conference on Adaptive Computing in Engineering Design and Control,* University of Plymouth, 97–102.

Powell, D. and M. M. Skolnick (1993). Using genetic algorithms in engineering design optimization with non-linear constraints. In S. Forrest (Ed.), *Proceedings of the* 5^{th} *International Conference on Genetic Al–gorithms,*Morgan Kaufmann, 424-430.

Rechenberg, I. (1973). *Evolutionstrategie: Optimierung Technisher Sys–teme nach Prinzipien des Biologischen Evolution.* Stuttgart: Fromman–Holzboog Verlag.

Reynolds, R. (1994). An introduction to cultural algorithms. In *Proceedings of the 3rd Annual Conference on Evolutionary Programming,* World Scientific, 131-139.

Reynolds, R., Z. Michalewicz, and M. Cavaretta (1995). Using cultural algorithms for constraint handling in Genocop. In J. R. McDonnell, R. G. Reynolds, and D. B. Fogel (Eds.), *Proceedings of the 4th Annual Conference on Evolutionary Programming,* MIT Press, 298-305.

Richardson, J. T., M. R. Palmer, G. Liepins, and M. Hilliard (1989). Some guidelines for genetic algorithms with penalty functions. In J. D. Schaffer (Ed.), *Proceedings of the 3rd International Conference on Genetic Algorithms,* Morgan Kaufmann, 191-197.

Schaffer, J.D., ed. (1989). *Proceedings of the 3rd International Conference on Genetic Algorithms.* Morgan Kaufmann, San Mateo, CA.

Schaffer, J.D. (1984). Some Experiments in Machine Learning Using Vector Evaluated Genetic Algorithms. PhD Dissertation, Vanderbilt University, Nashville, TN.

Schmidt, M. and Michalewicz, Z. (2000). Test–Case Generator TCG-2 for Nonlinear Parameter Optimization. In Proceedings of the 6th Parallel Problem Solving from Nature, Paris, September 17–20, 2000, Schoneauer, M., K. Deb, G. Rudolph, X. Yao, E. Lutton, J.J. Merelo, and H.-P. Schwefel (Editors), Springer-Verlag, Lecture Notes in Lomputer Science, Vol.1917, 539-548.

Schoenauer, M. and Z. Michalewicz (1996). Evolutionary Computation at the Edge of Feasibility. In *Proceedings of the 4th Conference on Parallel Problem Solving from Nature,* Voigt, H.–M., W. Ebeling, I. Rechenberg, and H.–P. Schwefel,(eds.), Lecture Notes in Computer Science, Vol.1141, Springer, Berlin, 245-254.

Schoenauer, M. and S. Xanthakis (1993). Constrained GA optimization. In S. Forrest (Ed.), *Proceedings of the 5th International Conference on Genetic Algorithms,* Morgan Kaufmann, 573–580.

Smith, A. and D. Tate (1993). Genetic optimization using a penalty function. In S. Forrest (Ed.), *Proceedings of the 5th International Conference on Genetic Algorithms,* Morgan Kaufmann, 499–503.

Surry, P., N. Radcliffe, and I. Boyd (1995). A multi-objective approach to constrained optimization of gas supply networks. In T. Fogarty (Ed.), *Proceedings of the AISB–95 Workshop on Evolutionary Computing,* Volume 993, Springer Verlag, 166–180.

Waagen, D., P. Diercks, and J. McDonnell (1992). The stochastic direction set algorithm: A hybrid technique for finding function extrema. In

D. B. Fogel and W. Atmar (Eds.), *Proceedings of the 1st Annual Conference on Evolutionary Programmming,*Evolutionary Programming Society, 35–42.

Whitley, D., V.S. Gordon, and K. Mathias (1996). Lamarckian Evolution, the Baldwin Effect and Function Optimization. In *Proceedings of the 3rd Conference on Parallel Problem Solving from Nature,* Davidor, Y., H.-P. Schwefel, and R. Männer, (eds.), Lecture Notes in Computer Science, Vol.866, Springer, Berlin, 6-15.

Whitley, D., K. Mathias, S. Rana, and J. Dzubera (1995). Building better test functions. In *Proceedings of the Sixth International Conference on Genetic Algorithms,* Eshelman, L.J.(ed.), Morgan Kaufmann, San Mateo, CA.

Whitley, D., K. Mathias, S. Rana, and J. Dzubera (1996). Evaluating evolutionary algorithms. Artificial Intelligence Journal, 85, 245–276.

Xiao, J., Z. Michalewicz, L. Zhang, and K. Trojanowski (1997). Adaptive Evolutionary Planner/Navigator for Mobile Robots. *IEEE Transactions on Evolutionary Computation,* 1(1), 18-28.

Chapter 4

CONSTRAINED EVOLUTIONARY OPTIMIZATION

— the penalty function approach

Thomas Runarsson and
Xin Yao

Abstract The penalty function method has been used widely in constrained evolutionary optimization (CEO). This chapter provides an in-depth analysis of the penalty function method from the point of view of search landscape transformation. The analysis leads to the insight that applying different penalty function methods in evolutionary optimization is equivalent to using different selection schemes. Based on this insight, two constraint handling techniques, i.e., stochastic ranking and global competitive ranking, are proposed as selection schemes in CEO. Our experimental results have shown that both techniques performed very well on a set of benchmark functions. Further analysis of the two techniques explains why they are effective: they introduce few local optima except for those defined by the objective functions.

Keywords: Constrained evolutionary optimization (CEO), penalty function method, ranking.

1. Introduction

The general nonlinear programming problem can be formulated as

$$\text{minimize} \quad f(\mathbf{x}), \quad \mathbf{x} = (x_1, \ldots, x_n) \in \mathcal{R}^n \tag{4.1}$$

where $f(\mathbf{x})$ is the objective function, $\mathbf{x} \in \mathcal{S} \cap \mathcal{F}$, $\mathcal{S} \subseteq \mathcal{R}^n$ defines the *search space* which is an n-dimensional space bounded by the *parametric constraints*

$$\underline{x}_j \leq x_j \leq \overline{x}_j, \quad j \in \{1, \ldots, n\}, \tag{4.2}$$

and the *feasible region* $\mathcal{F}$ is defined by

$$\mathcal{F} = \{\mathbf{x} \in \mathcal{R}^n \mid g_k(\mathbf{x}) \leq 0 \; \forall \; k \in \{1, \ldots, m\}\}, \tag{4.3}$$

where $g_k(\mathbf{x}), k \in \{1, \ldots, m\}$, are inequality *constraints*. Equality constraints $h(\mathbf{x})$ can be approximated by inequality constraints using $|h(\mathbf{x})| - \delta \leq 0$, where δ is a small positive number that indicates the degree of constraint violation. Only minimization problems are considered in this chapter without loss of generality since $\max\{f(\mathbf{x})\} = -\min\{-f(\mathbf{x})\}$.

The penalty function methods considered in this chapter belong to the exterior penalty approach. They are used widely in evolutionary constrained optimization (ECO), although some of the methods are equally applicable to non-evolutionary optimization algorithms. In contrast to numerous penalty function methods proposed for ECO (Michalewicz and Schoenauer, 1996), few theoretical analysis are available to explain how and why a penalty function method works. This chapter fills in this gap by providing an in-depth analysis of penalty function methods and their relationship to search landscape transformation. Such analysis has led to the development of new constraint handling techniques for CEO. In essence, a penalty function method transforms the search landscape by adding a penalty term to the objective function. Such transformation influences the relative fitness of individuals in a population. It also alters the characteristics of the search landscape, e.g., ruggedness. A previously fit individual according to the objective function might not be fit anymore on the transformed search landscape. Since the primary, if not the only, place in an evolutionary algorithm that fitness is used is selection, it is easy to see that an effective approach to "implementing" a penalty function method is to design a new selection scheme. Two rank-based selection schemes are described in this chapter to illustrate how penalty function methods can be "implemented" effectively by designing new ranking schemes in ECO.

The rest of this chapter is organized as follows. Section 2 analysis the penalty function method in CEO and discusses how different penalty function methods influence evolutionary search. In particular, the relationship between different penalty function methods and the ranking of individuals in a population is discussed in detail. Sections 3 and 4 present the ideas and algorithms of two constraint handling techniques based on ranking, i.e., stochastic ranking (Runarsson and Yao, 2000) and global competitive ranking. Section 5 provides further analysis of penalty function methods and shows how the penalty function method works through two detailed examples. Section 6 gives our experimental results on the two constraint handling techniques. Finally, Section 7 gives a brief summary of this chapter.

2. The Penalty Function Method

Constrained optimization problems have often been transformed into unconstrained ones by adding a measure of the constraint violation to the objective function (Fiacco and McCormick, 1968). This constrained handling technique is known as the penalty function method.

The introduction of the penalty term enables the transformation of a constrained optimization problem into a series of unconstrained ones, e.g.,

$$\psi(\mathbf{x}) = f(\mathbf{x}) + r^{(g)}\ \phi(\mathrm{g}_j(\mathbf{x}); j = 1, \ldots, m) \tag{4.4}$$

where $\phi \geq 0$ is a real valued function which imposes a penalty, $\phi(\mathrm{g}_j(\mathbf{x}))$, controlled by a sequence of *penalty coefficients* $\{r^{(g)}\}_0^G$. G indicates the maximum number of generations used in CEO. The general form of function ϕ includes both the generation counter g (for dynamic penalty) and the population (for adaptive penalty). In our current notation, this is reflected in the penalty coefficient $r^{(g)}$. This transformation, i.e. equation (4.4), has been used widely in CEO (Kazarlis and Petridis, 1998; Siedlecki and Sklansky, 1989). In particular, the following quadratic loss function (Fiacco and McCormick, 1968), whose decrease in value represents an approach to the feasible region, has often been used as the *penalty function* (Michalewicz and Attia, 1994; Joines and Houck, 1994):

$$\phi(\mathrm{g}_j(\mathbf{x}); j = 1, \ldots, m) = \sum_{j=1}^{m} \max\{0, \mathrm{g}_j(\mathbf{x})\}^2. \tag{4.5}$$

However, any other penalty function is equally valid. Different penalty functions characterize different problems. It is unlikely that a generic penalty function exists which is optimal for all problems. The introduction of penalties may transform a smooth objective function into a rugged one. The search may then become more easily trapped in local minima. For this reason, it is necessary to develop a penalty function method which attempts to preserve the topology of the objective function and yet enables a CEO algorithm to locate the optimal feasible solution.

The penalty function method may work quite well for some problems. However, deciding an optimal (or near-optimal) value for $r^{(g)}$ turns out to be a difficult optimization problem itself! If $r^{(g)}$ is too small, an infeasible solution may not be penalized enough. Hence an infeasible solution may be evolved by an evolutionary algorithm. If $r^{(g)}$ is too large, then a feasible solution is very likely to be found but could be of very poor quality. A large $r^{(g)}$ discourages the exploration of infeasible regions even in the early stages of evolution. This is particularly ineffective for problems where feasible regions in the whole search space are

disjoint. In this case, it may be difficult for an evolutionary algorithm to move from one feasible region to another unless they are very close to each other. Reasonable exploration of infeasible regions may act as bridges connecting two or more different feasible regions. The critical issue here is how much exploration of infeasible regions (i.e., how large $r^{(g)}$ is) should be considered as reasonable. The answer to this question is problem dependent. Even for the same problem, different stages of evolutionary search may require different $r^{(g)}$ values.

There has been some work on the dynamic setting of $r^{(g)}$ values in CEO (Joines and Houck, 1994; Kazarlis and Petridis, 1998; Michalewicz and Attia, 1994). Such work usually relies on a predefined monotonically nondecreasing sequence of $r^{(g)}$ values. This approach worked well for some simple problems but failed for more difficult ones because the optimal setting of $r^{(g)}$ values is problem dependent (Reeves, 1997). A fixed and predefined sequence cannot solve a variety of different problems satisfactorily. A trial-and-error process has to be used in this situation in order to find a proper function for $r^{(g)}$ for each problem, as is done in (Joines and Houck, 1994; Kazarlis and Petridis, 1998).

An adaptive approach, where $r^{(g)}$ values are adjusted dynamically and automatically by an evolutionary algorithm itself, appears to be most promising in tackling different constrained optimization problems. For example, population information can be used to adjust $r^{(g)}$ values adaptively (Smith and Coit, 1997). Different problems lead to different populations in evolutionary search and thus lead to different $r^{(g)}$ values. The advantage of such an adaptive approach is that it can be applied to problems where little prior knowledge is available because there is no need to find a predefined $r^{(g)}$ value, or a sequence of $r^{(g)}$ values.

According to (4.4), different $r^{(g)}$ values lead to different fitness functions. A fit individual under one fitness function may not be fit under a different fitness function. When rank-based selection is used in CEO, finding a near optimal $r^{(g)}$, adaptively, is equivalent to ranking individuals in a population adaptively. Hence, the issue of setting $r^{(g)}$ adaptively becomes how to rank individuals according to their objective and penalty values.

To facilitate later discussion, some notations are first introduced here. The individuals being ranked will be *arbitrarily* assigned some numerical labels. Then for any ranking of individuals, the corresponding permutation $\pi \in \mathcal{P}^{\lambda}$ will be a function from $\{1, \ldots, \lambda\}$ onto itself, whose arguments are the individuals and whose values are the ranks. The following notation is used: $\pi(i)$ is the rank given to individual i and $\pi^{-1}(j)$ is the individual assigned the rank j. Since $\pi^{-1}(j)$ is the individual assigned

the rank j, the bracket notation

$$\pi = \langle \pi^{-1}(1), \pi^{-1}(2), \ldots, \pi^{-1}(\lambda) \rangle$$

corresponds to listing all individuals in their ranked order.

For a given penalty coefficient $r^{(g)} > 0$ let the ranking of λ individuals be

$$\psi(\mathbf{x}_{\pi^{-1}(1)}) \leq \psi(\mathbf{x}_{\pi^{-1}(2)}) \leq \ldots \leq \psi(\mathbf{x}_{\pi^{-1}(\lambda)}) \tag{4.6}$$

where ψ is the transformation function given by equation (4.4). Now examine the adjacent pair $\pi^{-1}(i)$ and $\pi^{-1}(i+1)$ in the ranked order:

$$f_i + r^{(g)}\phi_i \leq f_{i+1} + r^{(g)}\phi_{i+1}, \qquad i \in \{1, \ldots, \lambda - 1\}, \tag{4.7}$$

where notations $f_i = f(\mathbf{x}_{\pi^{-1}(i)})$ and $\phi_i = \phi(\mathrm{g}_j(\mathbf{x}_{\pi^{-1}(i)}), j = 1, \ldots, m))$ are used for convenience.

Define a parameter, $\check{r}_i$, which will be referred to as the *critical penalty coefficient* for the adjacent pair i and $i+1$, as

$$\check{r}_i = (f_{i+1} - f_i)/(\phi_i - \phi_{i+1}), \qquad \text{for } \phi_i \neq \phi_{i+1}. \tag{4.8}$$

For a given choice of $r^{(g)} \geq 0$, there are three different cases which may give rise to Inequality (4.7):

1 $f_i \leq f_{i+1}$ and $\phi_i \geq \phi_{i+1}$: the comparison is said to be *dominated by the objective function* and $0 < r^{(g)} \leq \check{r}_i$ because the objective function f plays the dominant role in determining the inequality. When individuals are feasible, $\phi_i = \phi_{i+1} = 0$ and $\check{r}_i \to \infty$.

2 $f_i \geq f_{i+1}$ and $\phi_i < \phi_{i+1}$: the comparison is said to be *dominated by the penalty function* and $0 < \check{r}_i < r^{(g)}$ because the penalty function ϕ plays the dominant role in determining the inequality.

3 $f_i < f_{i+1}$ and $\phi_i < \phi_{i+1}$: the comparison is said to be *nondominated* and $\check{r}_i < 0$.

When comparing nondominated and feasible individuals, the value of $r^{(g)}$ has no impact on Inequality (4.7). In other words, it does not change the order of ranking of the two individuals. However, the value of $r^{(g)}$ is critical in the first two cases as $\check{r}_i$ is the flipping point that will determine whether the comparison is objective or penalty function dominated. For example, if $r^{(g)}$ is increased to a value greater than $\check{r}_i$ in the first case, individual $\pi^{-1}(i+1)$ would change from a fitter individual into a less fit one. For the entire population, the chosen value of $r^{(g)}$ used for comparisons will determine the fraction of individuals dominated by the objective and penalty functions.

Not all possible $r^{(g)}$ values can influence the ranking of individuals. They have to be within a certain range, i.e. $\underline{r}_g < r^{(g)} < \overline{r}_g$, to influence the ranking, where the lower bound $\underline{r}_g$ is the minimum critical penalty coefficient computed from adjacent individuals ranked only according to the objective function and the upper bound $\overline{r}_g$ is the maximum critical penalty coefficient computed from adjacent individuals ranked only according to the penalty function. In general, there are three different categories of $r^{(g)}$ values (Runarsson and Yao, 2000):

1 $r^{(g)} < \underline{r}^{(g)}$: *All* comparisons are based only on the objective function. $r^{(g)}$ is too small to influence the ranking of individuals. This is called *under-penalization.*

2 $r^{(g)} > \overline{r}^{(g)}$: *All* comparisons are based only on the penalty function. $r^{(g)}$ is so large that the impact of the objective function can be ignored. This is called *over-penalization.*

3 $\underline{r}^{(g)} < r^{(g)} < \overline{r}^{(g)}$: *All* comparisons are based on a combination of objective and penalty functions.

Penalty function methods can be classified into one of the above three categories. Some methods may fall into different categories during different stages in evolutionary search. It is important to understand the difference among these three categories because they indicate which function (or combination of functions) is driving the search process and how search progresses. For example, most dynamic penalty methods start with a low $r^{(g)}$ value (i.e., $r^{(g)} < \underline{r}^{(g)}$) in order to find a good region that may contain both feasible and infeasible individuals. Towards the end of search, a high $r^{(g)}$ value (i.e., $r^{(g)} > \overline{r}^{(g)}$) is often used in order to locate a good feasible individual. Such a dynamic penalty method would work well for problems for which the unconstrained global optimum is close to the constrained global optimum. It is unlikely to work well for problems for which the constrained global optimum is far away from the unconstrained one, because the initial low $r^{(g)}$ value would drive the search towards the unconstrained global optimum and thus further away from the constrained one.

The traditional constraint handling technique used in evolution strategies (ESs) falls roughly into the category of over-penalization since all infeasible individuals are regarded as worse than feasible ones (Schwefel, 1995; Hoffmeister and Sprave, 1996; Deb, 1999; Jiménez and Verdegay, 1999). In fact, canonical evolution strategies allow only feasible individuals in the initial population. To perform constrained optimization, an ES is first used to find a feasible initial population by minimizing the penalty function (Schwefel, 1995, p. 115). Once a feasible initial

population is found, the ES algorithm will then minimize the objective function and reject all infeasible solutions generated.

It has been widely recognized that neither under- nor over-penalization is a good constraint handling technique and there should be a balance between preserving feasible individuals and rejecting infeasible ones (Gen and Cheng, 1997; Runarsson and Yao, 2000). Such a balance can be achieved by adjusting our measure of how fit an individual should be in comparison with others. The adjustment can be done explicitly through ranking individuals in evolutionary algorithms. In order to strike the right balance, ranking should be dominated by a mixture of objective and penalty functions. That is, the penalty coefficient $r^{(g)}$ should be within the bounds: $\underline{r}^{(g)} < r^{(g)} < \overline{r}^{(g)}$. It is worth noting that the two bounds are not fixed. They are problem dependent and may change from generation to generation as they are also influenced by the current population.

One way to measure the balance of dominance of objective and penalty functions is to count how many comparisons of adjacent individual pairs are dominated by the objective and penalty functions respectively. Such a number of comparisons can be computed for any given $r^{(g)}$ by counting the number of critical penalty coefficients given by (4.8) which are greater than $r^{(g)}$. If there is a predetermined preference for the number of adjacent comparisons that should be dominated by the penalty function then a corresponding penalty coefficient can be determined.

It is clear from the analysis in this section that all a penalty function method tries to do is to obtain the right balance between objective and penalty functions so that the search moves towards the optimal feasible solution rather than the optimum in the combined feasible and infeasible space. One way to achieve such balance effectively and efficiently is to adjust such balance directly and explicitly. Possible methods of achieving this will be presented in the following two sections.

3. Stochastic Ranking

The ranking procedure introduced in this section is *stochastic ranking* (Runarsson and Yao, 2000) where ranking is achieved by a bubble-sort-like procedure. In this approach a probability P_f of using only the objective function for comparing individuals in the infeasible region of the search space is introduced. That is, given any pair of two adjacent individuals, the probability of comparing them (in order to determine which

one is fitter) according to the objective function is 1 if both individuals are feasible, otherwise it is P_f.

The procedure provides a convenient way of balancing the dominance in a ranked set. In the bubble-sort-like procedure, λ individuals are ranked by comparing adjacent individuals in at least λ sweeps[1]. The procedure is halted when no change in the rank ordering occurs within a complete sweep. Figure 4.1 shows the stochastic bubble sort procedure used to rank individuals in a population (Runarsson and Yao, 2000).

If at least one individual is infeasible in an adjacent pair, the probability of an individual winning a comparison, i.e., holding the higher rank, in the ranking procedure is

$$P_w = P_{fw}P_f + P_{\phi w}(1 - P_f) \tag{4.9}$$

where P_{fw} is the probability of the individual winning according to the objective function and $P_{\phi w}$ is the probability of the individual winning

[1]It would be exactly λ sweeps if the comparisons were not made stochastic.

Stochastic bubble sort (P_f, f, ϕ):
  $\pi(j) = j \ \forall \ j \in \{1, \ldots, \lambda\}$;
  for $i = 1$ to N **do**
    for $j = 1$ to $\lambda - 1$ **do**
      sample $u \in U(0, 1)$;
      if $(\phi(\mathbf{x}_{\pi^{-1}(j)}) = \phi(\mathbf{x}_{\pi^{-1}(j+1)}) = 0)$ **or** $(u < P_f)$ **then**
        if $(f(\mathbf{x}_{\pi^{-1}(j)}) > f(\mathbf{x}_{\pi^{-1}(j+1)}))$ **then**
          $swap(\pi^{-1}(j), \pi^{-1}(j+1))$;
        fi
      else
        if $(\phi(\mathbf{x}_{\pi^{-1}(j)}) > \phi(\mathbf{x}_{\pi^{-1}(j+1)}))$ **then**
          $swap(\pi^{-1}(j), \pi^{-1}(j+1))$;
        fi
      fi
    od
    if no *swap* done **break**; **fi**
  od
return (π)

Figure 4.1. Stochastic ranking procedure, where $U(0, 1)$ is a uniform random number generator and N is the number of sweeps going through the whole population. When $P_f = 0$ the ranking is equivalent to over-penalization. When $P_f = 1$ the ranking is equivalent to under-penalization. The initial ranking is generated at random.

according to the penalty function. In the case where adjacent individuals are both feasible $P_w = P_{fw}$, the probability of winning k more comparisons than losses can be computed. The total number of wins will be $k' = (N + k)/2$ where N is the total number of comparisons made. The probability of winning k' comparisons out of N is given by the binomial distribution

$$P_w(y = k') = \binom{N}{k'} P_w^{k'}(1 - P_w)^{N-k'}. \tag{4.10}$$

The probability of winning *at least* k' comparisons is

$$P'_w(y \geq k') = 1 - \sum_{j=0}^{k'-1} \binom{N}{j} P_w^{j}(1 - P_w)^{N-j}. \tag{4.11}$$

Equations (4.10) and (4.11) show that the greater the number of comparisons (N) the less influence the initial ranking will have. It is worth noting that the probability P_w usually varies for different individuals in different stages of ranking. Now consider the case where P_w is constant during the entire ranking procedure, which will be true if $f_i < f_j$, $\phi_i > \phi_j$; $j \neq i, j = 1, \ldots, \lambda$. Then $P_{fw} = 1$ and $P_{\phi w} = 0$. If $P_f = 0.5$ is chosen then $P_w = 0.5$. There will be an equal chance for a comparison to be made based on the objective or penalty function. Because we are interested in feasible solutions as the final solution, P_f should be less than 0.5 such that there is a pressure against infeasible solutions. The strength of the pressure can be adjusted easily by adjusting only P_f. When parameter N, the number of sweeps, approaches ∞, the ranking will be determined by P_f. That is, if $P_f > 0.5$, the ranking will be based on the objective function. If $P_f < 0.5$, the ranking is equivalent to over-penalization. Hence, an increase in the number of ranking sweeps is effectively equivalent to changing parameter P_f. Hence, $N = \lambda$ can be fixed and P_f adjusted to achieve the best performance.

The effectiveness and efficiency of stochastic ranking will be evaluated in Section 6 through experimental studies.

4. Global Competitive Ranking

A different method of ranking individuals in a population, in order to strike the right balance between objective and penalty functions, is the deterministic global competitive ranking scheme. In this scheme, an individual i is ranked by comparing it against all other members of the population. This is different from the stochastic ranking approach where only adjacent individuals compete for a given rank. In the global

competitive ranking method, special consideration is given to *tied ranks*. In the case of tied ranks the same lower rank will be used. For example, for ranking $\pi = \langle 1, 3, (2,6), 7, (4,5)\rangle$, we should have $\pi(1) = 1$, $\pi(3) = 2$, $\pi(2) = \pi(6) = 3$, $\pi(7) = 5$ and $\pi(4) = \pi(5) = 6$.

Similar to the stochastic ranking approach, it is assumed that either the objective or the penalty function will be used in deciding an individual's rank. P_f indicates the probability that a comparison is done based on the objective function only. The probability that individual i holds its rank $\pi(i)$ when challenged by any other member of the population is,

$$P(\pi(i)) = P_f \frac{\lambda - \pi_f(i)}{\lambda - 1} + (1 - P_f)\frac{\lambda - \pi_\phi(i)}{\lambda - 1}, \tag{4.12}$$

where the permutations π_f and π_ϕ correspond to the ranking of individuals based on the objective and penalty functions, respectively. Equation (4.12) can be used to determine the final ranking. That is, the fitness function for the minimization problem becomes:

$$\psi(\mathbf{x}_i) = P_f \frac{\pi_f(i) - 1}{\lambda - 1} + (1 - P_f)\frac{\pi_\phi(i) - 1}{\lambda - 1}. \tag{4.13}$$

It is clear from the above that P_f can be used easily to bias ranking according to the objective or penalty function. In practice, the probability should take a value $0 < P_f < 0.5$ in order to guarantee that a feasible solution may be found. The close the probability is to 0.5, the greater the emphasis will be on minimizing the objective function. As the P_f approaches 0, not equal to zero, the ranking corresponds an over-penalization. The global competitive ranking scheme, unlike stochastic ranking, is deterministic. It can be summarized by Figure 4.2.

Global competitive ranking (P_f, f, ϕ):
Step 1: Determine the ranking, π_f, π_ϕ:
$f(\mathbf{x}_{\pi_f^{-1}(1)}) \leq f(\mathbf{x}_{\pi_f^{-1}(2)}) \leq \ldots \leq f(\mathbf{x}_{\pi_f^{-1}(\lambda)})$
$\phi(\mathbf{x}_{\pi_\phi^{-1}(1)}) \leq \phi(\mathbf{x}_{\pi_\phi^{-1}(2)}) \leq \ldots \leq \phi(\mathbf{x}_{\pi_\phi^{-1}(\lambda)})$
Step 2. Compute competitive fitness:
$\psi(\mathbf{x}_i) = P_f \frac{\pi_f(i)-1}{\lambda-1} + (1 - P_f)\frac{\pi_\phi(i)-1}{\lambda-1}$.
Step 3. Determine final ranking, π:
$\psi(\mathbf{x}_{\pi^{-1}(1)}) \leq \psi(\mathbf{x}_{\pi^{-1}(2)}) \leq \ldots \leq \psi(\mathbf{x}_{\pi^{-1}(\lambda)})$

Figure 4.2. Global competitive ranking method for constraint handling.

5. How Penalty Methods Work

Convergence and convergence rate are two important issues in stochastic optimization and search algorithms, such as EAs. For a stochastic search procedure, average positive progress towards the global optimum, $\mathbf{x}^*$, is necessary in order to find the optimum efficiently. One approach of measuring progress is to compute the distance travelled between successive generations (Schwefel, 1995) towards $\mathbf{x}^*$. The distance from the best individual in generation (g) to the optimum $\mathbf{x}^*$ should be on average greater than that of the best individual at generation $(g+1)$. That is, the following $\varphi_{\mathbf{x}}$ should be greater than 0:

$$\varphi_{\mathbf{x}} = \mathrm{E}\left[d\left(\mathbf{x}^{(g)}_{\pi^{-1}(1)}, \mathbf{x}^*\right) - d\left(\mathbf{x}^{(g+1)}_{\pi^{-1}(1)}, \mathbf{x}^*\right) \middle| \mathbf{x}^{(g)}_{\pi^{-1}(1)}, \ldots, \mathbf{x}^{(g)}_{\pi^{-1}(\mu)}\right], \tag{4.14}$$

where the distance metric $d(\mathbf{x}, \mathbf{x}^*) = \| \mathbf{x} - \mathbf{x}^* \|$. A similar progress definition is given by (Rudolph, 1997, p. 207) in terms of fitness for the unconstrained problem:

$$\varphi_f = \mathrm{E}\left[f\left(\mathbf{x}^{(g)}_{\pi^{-1}(1)}\right) - f\left(\mathbf{x}^{(g+1)}_{\pi^{-1}(1)}\right) \middle| \mathbf{x}^{(g)}_{\pi^{-1}(1)}, \ldots, \mathbf{x}^{(g)}_{\pi^{-1}(\mu)}\right]. \tag{4.15}$$

However, the progress rate computed from fitness values, as the one given by (4.15), indicates the progress towards a local unconstrained minimum only. Progress towards the global minimum in a multimodal landscape can only be computed in terms of the distance and when the global minimum is known (Yao et al., 1999). Computing φ analytically is a difficult theoretical problem although there has been some published work on drift analysis (He and Yao, 2001).

If positive progress towards the global optimum is to be maintained, there must exist at least one parent $\mathbf{x}^{(g)}$ which produces at least one offspring that is closer than itself to the optimum $\mathbf{x}^*$ on average. Consider a simple $(1, \lambda)$ EA where there is only one parent $(\mu = 1)$ at each generation producing λ offspring. The offspring are produced using the following mutation operator:

$$\mathbf{x}^{(g+1)}_{\pi^{-1}(i)} = \mathbf{x}^{(g)}_{\pi^{-1}(1)} + N_i(0, \sigma^2) \quad i = 1, \ldots, \lambda, \tag{4.16}$$

where $N_i(0, \sigma^2)$ is a normally distributed random variable with zero mean and variance σ^2. We can now use two examples to illustrate how a penalty function method works by investigating the relationship between different penalty function methods and progress rates. In particular, we will examine how the progress in terms of fitness corresponds to that in terms of the distance to the global optimum.

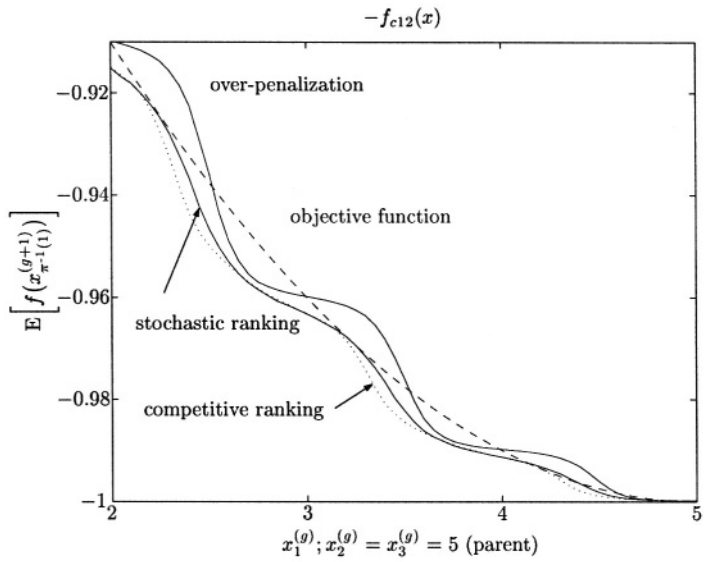


Figure 4.3. Expected fitness of the best offspring as a function of parent position for test function f_{12}. The curves lying below the dashed one (parent fitness) corresponds to positive progress towards the global optimum.

The first example is a the benchmark test function, f_{12} in (Koziel and Michalewicz, 1999):

$$\text{maximize: } f_{12}(\mathbf{x}) = \left(100 - (x_1 - 5)^2 - (x_2 - 5)^2 - (x_3 - 5)^2\right)/100$$

subject to:

$$\mathrm{g}(\mathbf{x}) = (x_1 - p)^2 + (x_2 - q)^2 + (x_3 - r)^2 - 0.0625 \leq 0,$$

where $0 \leq x_i \leq 10$ $(i = 1, 2, 3)$ and $p, q, r = 1, 2, \ldots, 9$. The feasible region of the search space consists of 9^3 disjointed spheres. A point (x_1, x_2, x_3) is feasible if and only if there exist p, q, r such that the above inequality holds. Hence, the $\mathrm{g}(\mathbf{x})$ returned corresponds to its lowest value for given p, q, r values. The feasible global optimum is located at $\mathbf{x}^* = (5,5,5)$ where $f_{12}(\mathbf{x}^*) = 1$.

Figure 4.3 shows the results of 10,000 one-generational experiments for a number of different parent values. In Figure 4.3, variables x_2 and x_3 were fixed at 5 and only x_1 was adjusted between values 2 and 5. The mean search step size used was $\sigma = 0.2$ and the number of offspring $\lambda = 10$. This simulation was conducted using three different ranking strategies: *over-penalization, stochastic ranking,* and *global competitive ranking.* In both the stochastic and global competitive ranking, the value

of P_f is 0.45. Over-penalization corresponds to a ranking with $P_f = 0$. The problem was treated as a minimization one.

In Figure 4.3, the expected objective function value of the highest ranked offspring is plotted versus the parent value of x_1. The dashed line corresponds to the objective function value of the parent. Hence, positive progress toward the global optimum will be achieved when the expected objective function value of the best offspring lies beneath the dashed line. The figure illustrates how the over-penalization approach has effectively transformed the original unimodal objective function to a multimodal fitness function. There existed large regions of negative progress when the over-penalization approach was used. The stochastic and global competitive ranking, however, maintained their positive progress towards the global feasible optimum even in infeasible regions, although the rate of progress is slower. This example shows that the penalty function method works by transforming the search landscape (Runarsson, 2000). Inappropriate penalty functions may make the optimization task more difficult than it should be.

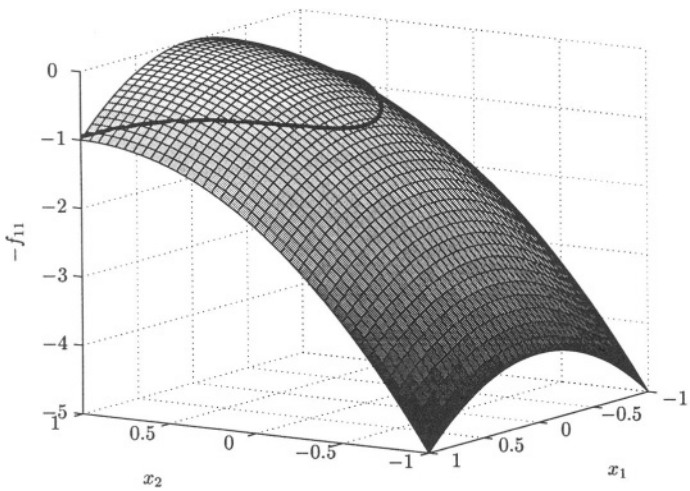


Figure 4.4. Fitness landscape for test function f_{11}. The curve represents the region of feasibility.

The second example is also a well known benchmark test function in (Koziel and Michalewicz, 1999):

$$\text{minimize: } f_{11}(\mathbf{x}) = x_1^2 + (x_2 - 1)^2$$

subject to:

$$\mathrm{h}(\mathbf{x}) = x_2 - x_1^2 = 0,$$

where $-1 \leq x_1 \leq 1$ and $-1 \leq x_2 \leq 1$. The global feasible optimum is at $\mathbf{x}^* = (\pm 1/\sqrt{2}, 1/2)$ where $f_{11}(\mathbf{x}^*) = 0.75$. Figure 4.4 shows the objective function, $f_{11}(\mathbf{x})$, and the constraint curve $\mathrm{h}(\mathbf{x})$.

In this example both parent variables x_1 and x_2 were varied in our experimental study. *Stochastic ranking* ($P_f = 0.45$) was compared with *over-penalization* ($P_f = 0$). Since there exist two optima for this example, the progress was computed in terms of the maximum distance covered towards one of the optima:

$$\varphi_x = \mathrm{E}\Big[\min\left\{d\left(\mathbf{x}^{(g)}_{\pi^{-1}(1)}, \mathbf{y}^*\right), d\left(\mathbf{x}^{(g)}_{\pi^{-1}(1)}, \mathbf{z}^*\right)\right\} - \min\left\{d\left(\mathbf{x}^{(g+1)}_{\pi^{-1}(1)}, \mathbf{y}^*\right), d\left(\mathbf{x}^{(g+1)}_{\pi^{-1}(1)}, \mathbf{z}^*\right)\right\}\Big] \tag{4.17}$$

where $\mathbf{z}^*$ and $\mathbf{y}^*$ are the optima $(\pm 1/\sqrt{2}, 1/2)$.

Two different mean step sizes were used in our experiments: $\sigma = 0.05$ and $\sigma = 0.1$. The number of offspring generated was again $\lambda = 10$. The progress rate given by Equation 4.17 is illustrated by contour plots shown in Figure 4.5, where regions of negative progress are outlined with contour lines.

It is clear from Figure 4.5 that negative regions of progress were located around the global optima. This is not surprising since the mean search step size used was too large in these regions. A decreasing mean search step size should be used. For the over-penalization approach, however, there existed additional regions of negative progress which were not in the global optimum regions. These regions formed additional local attractors and would trap individuals as the mean search step size decreased. Stochastic ranking did not create any local attractors in this case. This is also true for global competitive ranking, as will be seen in the following section.

In summary, the introduction of constraints may produce additional local optima in the search landscape. A well designed constraint handling technique can minimize the number of such misleading local optima. This is the primary reason why our ranking methods worked so well on many test functions. Our ranking methods also make it easy to control constrained search by adjusting P_f for different problems.

6. Experimental Study

6.1 Evolutionary Optimization Algorithm

The evolutionary optimization algorithm described in this section is based on the evolution strategy (ES) (Schwefel, 1995). One reason for choosing ES is that it does not introduce any specialized constraint-handling variation operators. It will be shown that specialized and

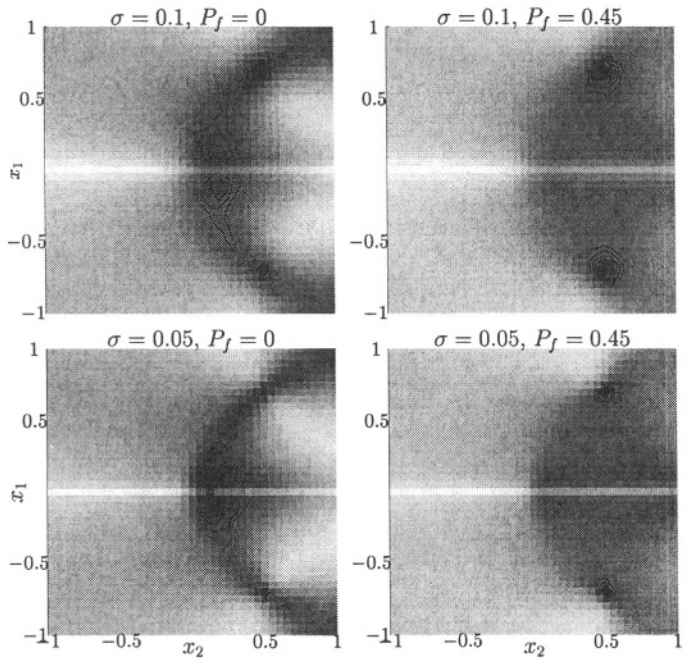


Figure 4.5. The figures show the progress rate in terms of the distance metric, i.e. φ_x where $\mu = 1$ and $\lambda = 10$, for test function f_{11}. The drawn contours mark regions of negative progress (darker regions). When $P_f = 0$ (over-penalization), there exists a region where no progress is maintained towards either global optima, and thus the search will get stuck in this region. This figure explains the poor performance observed in Table 4.1 for this function.

complex variation operators for constrained optimization problems are unnecessary although they may be quite useful for particular types of problems (see for example (Michalewicz et al., 1996)). A simple extension to the ES, i.e., the use of the ranking schemes proposed in the previous sections, can achieve significantly better results than other more complicated techniques.

In a (μ, λ)-ES algorithm, an individual i is a pair of real-valued vectors, $(\mathbf{x}_i, \sigma_i)$, $\forall\ i \in \{1, \ldots, \lambda\}$. The initial population of $\mathbf{x}$ is generated according to a uniform n-dimensional probability distribution over the search space $\mathcal{S}$. Let δx be an approximate measure of the expected distance to the global optimum, then the initial setting for the 'mean step sizes' should be (Schwefel, 1995, p. 117):

$$\sigma_{i,j}^{(0)} = \delta x_j / \sqrt{n} \approx (\overline{x}_j - \underline{x}_j)/\sqrt{n}, \quad i \in \{1, \ldots, \lambda\}, j \in \{1, \ldots, n\}, \quad (4.18)$$

where $\sigma_{i,j}$ denotes the j-th component of the vector σ_i. These initial values will also be used as upper bounds on σ.

Following the ranking schemes presented, the evaluated objective $f(\mathbf{x})$ and penalty function $\phi(g_k(\mathbf{x});\ k = 1, \ldots, m)$ for each individual $(\mathbf{x}_i, \sigma_i)$, $\forall\ i \in \{1, \ldots, \lambda\}$ is used to rank individuals in a population and the best (highest-ranked) μ individuals out of λ are selected for the next generation. The truncation level is set at $\mu/\lambda \approx 1/7$ (Bäck, 1996, p. 79).

Variation of strategy parameters is performed before the modification of objective variables. New λ strategy parameters are produced from the μ highest ranked individuals and then applied later for generating λ offspring. The 'mean step sizes' are updated according to the log-normal update rule (Schwefel, 1995): $i = 1, \ldots, \mu$, $h = 1, \ldots, \lambda$, and $j = 1, \ldots, n$,

$$\sigma_{h,j}^{(g+1)} = \hat{\sigma}_{h,j}^{(g)} \exp(\tau' N(0,1) + \tau N_j(0,1))), \tag{4.19}$$

where $N(0, 1)$ is a normally distributed one-dimensional random variable with an expectation of zero and variance one. The subscript j in $N_j(0, 1)$ indicates that the random number is generated anew for each value of j. The 'learning rates' τ and τ' are set equal to $\varphi^*/\sqrt{2\sqrt{n}}$ and $\varphi^*/\sqrt{2n}$ respectively where φ^* is the expected rate of convergence (Schwefel, 1995, p. 144) and is set to one (Bäck, 1996, p. 72). Recombination is performed on the self-adaptive parameters before applying the update rule given by (4.19). In particular, global intermediate recombination (the average) between two parents (Schwefel, 1995, p. 148) is implemented as

$$\hat{\sigma}_{h,j}^{(g)} = (\sigma_{i,j}^{(g)} + \sigma_{k_j,j}^{(g)})/2, \quad k_j \in \{1, \ldots, \mu\}, \tag{4.20}$$

where k_j is an index generated at random and anew for each j.

Having varied the strategy parameters, each individual $(\mathbf{x}_i, \sigma_i)$, $\forall\ i \in \{1, \ldots, \mu\}$, creates λ/μ offspring on average, so that a total of λ offspring are generated:

$$x_{h,j}^{(g+1)} = x_{i,j}^{(g)} + \sigma_{h,j}^{(g+1)} N_j(0,1) \tag{4.21}$$

Recombination is not used in the variation of objective variables. When an offspring is generated outside the parametric bounds defined by the problem, the mutation (variation) of the objective variable will be retried until the variable is within its bounds. In order to save computation time the mutation is retried only 10 times and then ignored, leaving the object variable in its original state within the parameter bounds.

Table 4.1. Over-penalization.

fcn	optimal	best	median	st. dev.	G_m
f_1	−15.000	−15.000	−15.000	0.0E+00	697
f_2	−0.803619	−0.803578	−0.785253	1.5E−02	1259
f_3	−1.000	−0.327	−0.090	7.2E−02	61
f_4	−30665.539	−30665.539	−30665.538	3.8E+00	632
f_5	5126.498	5126.945	5225.100	2.7E+02	213
f_6	−6961.814	−6961.814	−6961.814	1.9E−12	946
f_7	24.306	24.322	24.367	5.9E−02	546
f_8	−0.095825	−0.095825	−0.095825	2.7E−17	647
f_9	680.630	680.632	680.657	3.8E−02	414
f_{10}	7049.331	7117.416	7336.280	3.4E+02	530
f_{11}	0.750	0.750	0.953	5.4E−02	1750
f_{12}	−1.000000	−0.999972	−0.999758	1.4E−04	90
f_{13}	0.053950	0.919042	0.997912	1.5E−02	1750

6.2 Experimental Results and Discussion

Thirteen benchmark functions are studied. The first 12 are taken from (Koziel and Michalewicz, 1999) and the 13th from (Michalewicz, 1995). The details, including the original sources, of these functions are listed in appendix 4.A. Functions f_2, f_3, f_8, and f_{12} are maximization problems. They are transformed to minimization problems using $-f(\mathbf{x})$. For each of the benchmark problems 30 independent runs are performed using a (30, 200)-ES and the ranking procedures described in the previous sections. All runs are terminated after $G = 1750$ generations except for f_{12}, which was run for 175 generations. The experimental results using the stochastic and global competitive ranking, with $P_f = 0.45$, are given in Tables 4.2 to 4.3. The results are compared against the over-penalization approach (Table 4.1) used in ES (Hoffmeister and Sprave, 1996). The over-penalization approach corresponds to the ranking schemes discussed for $P_f \to 0$. In the tables the best feasible objective value, median, standard deviation, and median number of generations (G_m) needed to find the best individual are given.

As can be seen from Tables 4.1 to 4.3, both stochastic ranking and global competitive ranking performed very well for most test functions, especially for functions f_3, f_{11}, f_{12}, and f_{13}, for the reasons given in Section 5. They are also much faster than the over-penalization approach for most test functions. There are, however, two test functions that stand

Table 4.2. Stochastic ranking ($P_f = 0.45$).

fcn	optimal	best	median	st. dev.	G_m
f_1	−15.000	−15.000	−15.000	0.0E+00	741
f_2	−0.803619	−0.803515	−0.785800	2.0E−02	1086
f_3	−1.000	−1.000	−1.000	1.9E−04	1146
f_4	−30665.539	−30665.539	−30665.539	2.0E−05	441
f_5	5126.498	5126.497	5127.372	3.5E+00	258
f_6	−6961.814	−6961.814	−6961.814	1.6E+02	590
f_7	24.306	24.307	24.357	6.6E−02	715
f_8	−0.095825	−0.095825	−0.095825	2.6E−17	381
f_9	680.630	680.630	680.641	3.4E−02	557
f_{10}	7049.331	7054.316	7372.613	5.3E+02	642
f_{11}	0.750	0.750	0.750	8.0E−05	57
f_{12}	−1.000000	−1.000000	−1.000000	0.0E+00	82
f_{13}	0.053950	0.053957	0.057006	3.1E−02	349

Table 4.3. Global competitive ranking ($P_f = 0.45$).

fcn	optimal	best	median	st. dev.	G_m
f_1	−15.000	−15.000	−15.000	0.0E+00	692
f_2	−0.803619	−0.803591	−0.792805	1.7E−02	1335
f_3	−1.000	−1.000	−1.000	2.6E−05	1725
f_4	−30665.539	−30665.539	−30665.538	5.4E−01	731
f_5[2]	5126.498	5126.497	5126.721	1.1E+00	319
f_6	−6961.814	−6943.560	−6579.214	2.9E+02	13
f_7	24.306	24.308	24.361	1.1E−01	517
f_8	−0.095825	−0.095825	−0.095825	2.6E−17	398
f_9	680.630	680.631	680.657	5.8E−02	396
f_{10}	7049.331	–	–	–	–
f_{11}	0.750	0.750	0.750	7.2E−05	76
f_{12}	−1.000000	−1.000000	−1.000000	0.0E+00	63
f_{13}	0.053950	0.053943	0.053987	1.3E−04	247

out: f_{10} and f_6. It is difficult to determine whether it is the constraint handling technique or the underlying search method which is contributing to the success or failure in locating the optimum. In (Runarsson and Yao, 2000) the importance of the search method was illustrated on test

Table 4.4. Over-penalization versus stochastic ranking for test function f_{10} and $\varphi = 1/4$.

P_f	optimal	best	median	st. dev.	G_m
0.45	7049.331	7049.852	7054.111	5.7E+00	1733
0.00	7049.331	7049.955	7062.673	3.1E+01	1745

function f_{10} by setting $\varphi = 1/4$. This results is given in table 4.4 and illustrates how significant the search method is.

Test function f_6 is the only test function solved more effectively using over-penalization. For this reason it is interesting to plot its progress rate landscape. The test function has two variables. The progress rate is simulated as before using 10.000 one generational experiments in the region where suboptimal solutions are found. The result is depicted in figure 4.6. Progress landscapes for the step sizes $\sigma = 0.05$ (dotted) and $\sigma = 0.01$ (dashed) are plotted as contours. Negative progress is maintained to the right of the last of the three contour lines plotted. The solid lines in the figure are the constraint curves and the circle marks the location of $\mathbf{x}^*$. The feasible region is the top narrow band formed by the two constraint curves. From the figure it becomes clear that in this case over-penalization guides the search directly to the optimal feasible solutions from the infeasible region. However, stochastic ranking approaches the optimal solution from the combined feasible and infeasible region. The progress contours are simply rotated. In this test case no additional

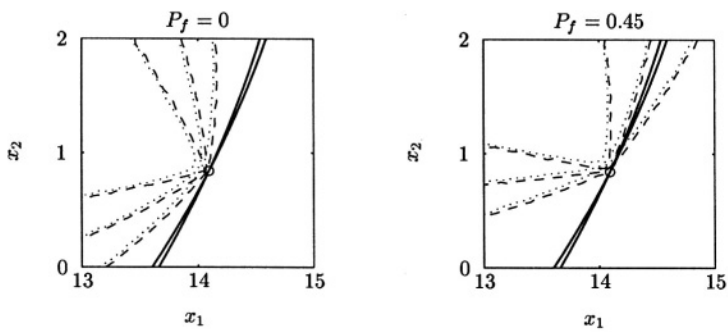


Figure 4.6. Progress landscape for test function f_6 for step sizes $\sigma = 0.05$ (dotted) and $\sigma = 0.01$ (dashed). Negative progress is to the right of the last of the three contour lines. The solid lines are the constraint curves and the circle the location of $\mathbf{x}^*$. The feasible region is the top narrow band formed by the two constraint curves.

attractors are created by the over-penalization method and therefore the two approaches should yield similar performance. This leads one to speculate whether the performance difference may be due to the lack of rotational invariance of the search method. To test this the coordinate system is rotated by $\pi/4$ and the experiment is re-run. The results are given in table 4.5. This simple experiment supports our prediction that the performance difference is due to the lack of rotational invariance of the search method.

Table 4.5. Over-penalization versus stochastic ranking for test function f_6 and coordinate system rotated by $\pi/4$.

P_f	best	median	mean	st. dev.	worst	G_m
0.45	−6954.352	−6913.419	−6909.142	2.7E+01	−6842.484	957
0.00	−6942.806	−6903.223	−6887.683	4.2E+01	−6782.945	864

7. Conclusion

The penalty function method is widely used in constrained optimization. It is emphasized in this chapter that the penalty function method transforms a constrained problem into an unconstrained one by modifying the search landscape. Different modifications lead to different search landscapes and thus different difficulties of optimization. We have given two concrete examples to illustrate how additional local optima could be introduced through inappropriate penalty methods and how such local optima could mislead search.

Selection in an EA depends primarily on fitness values of individuals. Modifications to a search (fitness) landscape can be achieved through modifications to the selection scheme, rather than to the fitness function. Ranking is a simple yet effective selection method that can be used to indicate which individuals are fitter than others and thus achieve the goal of modifying the fitness landscape. Two ranking schemes have been introduced in this paper to show how they can be used to handle constraints effectively and efficiently without adding a penalty term in the fitness function. Experimental results on a set of benchmark test functions are given in this chapter to support our analysis.

References

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

Deb, K. (1999). An efficient constrained handling method for genetic algorithms. In *Computer Methods in Applied Mechanics and Engineering,* page in press.

Fiacco, A. V. and McCormick, G. P. (1968). *Nonlinear Programming: Sequential Unconstrained Minimization Techniques.* Wiley, New-York.

Floundas, C. and Pardalos, P. (1987). *A Collection of Test Problems for Constrained Global Optimization,* volume 455 of *Lecture Notes in Computar Science.* Springer-Verlag, Berlin, Germany.

Gen, M. and Cheng, R. (1997). *Genetic Algorithms and Engineering Design.* Wiley, New-York.

He, J. and Yao, X. (2001). Drift analysis and average time complexity of evolutionary algorithms. *Artificial Intelligence,* 127(l):57–85.

Himmelblau, D. (1972). *Applied Nonlinear Programming.* McGraw-Hill, New-York.

Hock, W. and Schittkowski, K. (1981). *Test Examples for Nonlinear Programming Codes.* Lecture Notes in Economics and Mathematical Systems. Springer-Verlag, Berlin, Germany.

Hoffmeister, F. and Sprave, J. (1996). Problem independent handling of constraints by use of metric penalty functions. In Fogel, L. J., Angeline, P. J., and Bäck, T., editors, *Proceedings of the Fifth Annual Conference on Evolutionary Programming,* pages 289–294, Cambridge MA. The MIT Press.

Jiménez, F. and Verdegay, J. L. (1999). Evolutionary techniques for constrained optimization problems. In *Proc. of the 7th European Congress on Intelligent Techniques and Soft Computing (EUFIT'99),* Germany, Berlin. Springer-Verlag.

Joines, J. and Houck, C. (1994). On the use of non-stationary penalty functions to solve nonlinear constrained optimization problems with GAs. In *Proc. IEEE International Conference on Evolutionary Computation,* pages 579–584. IEEE Press.

Kazarlis, S. and Petridis, V. (1998). Varying fitness functions in genetic algorithms: Studying the rate of increase in the dynamic penalty terms. In *Parallel Problem Solving from Nature,* volume 1498 of *Lecture Notes in Computer Science,* pages 211–220, Berlin, Germany. Springer.

Koziel, S. and Michalewicz, Z. (1999). Evolutionary algorithms, homomorphous mappings, and constrained parameter optimization. *Evolutionary Computation,* 7(1): 19–44.

Michalewicz, Z. (1995). Genetic algorithms, numerical optimization and constraints. In Eshelman, L., editor, *Proceedings of the 6th International Conference on Genetic Algorithms,* pages 151–158, San Mateo, CA. Morgan Kaufman.

Michalewicz, Z. and Attia, N. (1994). Evolutionary optimization of constrained problems. In Fogel, L. J. and Sebald, A., editors, *Proc. of the 2nd Annual Conference on Evolutionary Programming,* pages 98–108, River Edge, NJ. World Scientific Publishing.

Michalewicz, Z., Nazhiyath, G., and Michalewicz, M. (1996). A note on usefulness of geometrical crossover for numerical optimization problems. In Fogel, L., Angeline, P., and Bäck, T., editors, *Proc. of the 5th Annual Conference on Evolutionary Programming,* pages 305–312. MIT Press, Cambridge, MA.

Michalewicz, Z. and Schoenauer, M. (1996). Evolutionary algorithms for constrained parameter optimization problems. *Evolutionary Computation,* 4(1):1–32.

Reeves, C. R. (1997). Genetic algorithms for the operations researcher. *INFORMS Journal on Computing,* 9(3):231–247.

Rudolph, G. (1997). *Convergence Properties of Evolutionary Algorithms.* Verlag Dr. Kovač, Hamburg.

Runarsson, T. P. (2000). *Evolutionary Problem Solving.* PhD thesis, University of Iceland, Reykjavik, Iceland.

Runarsson, T. P. and Yao, X. (2000). Stochastic ranking for constrained evolutionary optimization. *IEEE Transactions on Evolutionary Computation,* 4(3):284–294.

Schwefel, H.-P. (1995). *Evolution and Optimum Seeking.* Wiley, New-York.

Siedlecki, W. and Sklansky, J. (1989). Constrained genetic optimization via dynamic reward-penalty balancing and its use in pattern recognition. In *International Conference on Genetic Algorithms,* pages 141–149.

Smith, A. E. and Coit, D. W. (1997). Penalty functions. In Bäck, T., Fogel, D. B., and Michalewicz, Z., editors, *Handbook on Evolutionary Computation,* pages C5.2:l–6. Oxford University Press.

Yao, X., Liu, Y., and Lin, G. (1999). Evolutionary programming made faster. *IEEE Transactions on Evolutionary Computation,* 3(2):82–102.

Appendix: Test Function Suite

Minimize (Floundas and Pardalos, 1987):

$$f_1(\mathbf{x}) = 5\sum_{i=1}^{4} x_i - 5\sum_{i=1}^{4} x_i^2 - \sum_{i=5}^{13} x_i$$

subject to:

$$\begin{aligned}
g_1(\mathbf{x}) &= 2x_1 + 2x_2 + x_{10} + x_{11} - 10 \leq 0 \\
g_2(\mathbf{x}) &= 2x_1 + 2x_3 + x_{10} + x_{12} - 10 \leq 0 \\
g_3(\mathbf{x}) &= 2x_2 + 2x_3 + x_{11} + x_{12} - 10 \leq 0 \\
g_4(\mathbf{x}) &= -8x_1 + x_{10} \leq 0 \\
g_5(\mathbf{x}) &= -8x_2 + x_{11} \leq 0 \\
g_6(\mathbf{x}) &= -8x_3 + x_{12} \leq 0 \\
g_7(\mathbf{x}) &= -2x_4 - x_5 + x_{10} \leq 0 \\
g_8(\mathbf{x}) &= -2x_6 - x_7 + x_{11} \leq 0 \\
g_9(\mathbf{x}) &= -2x_8 - x_9 + x_{12} \leq 0
\end{aligned}$$

where the bounds are $0 \leq x_i \leq 1$ $(i = 1, \ldots, 9)$, $0 \leq x_i \leq 100$ $(i = 10, 11, 12)$ and $0 \leq x_{13} \leq 1$. The global minimum is at $\mathbf{x}^*$ = (1,1,1,1,1,1,1,1,1,3,3,3,1) where six constraints are active (g_1, g_2, g_3, g_7, g_8 and g_9) and $f_1(\mathbf{x}^*) = -15$.

Maximize (Koziel and Michalewicz, 1999):

$$f_2(\mathbf{x}) = \left| \frac{\sum_{i=1}^{n} \cos^4(x_i) - 2\prod_{i=1}^{n} \cos^2(x_i)}{\sqrt{\sum_{i=1}^{n} i x_i^2}} \right|$$

subject to:

$$g_1(\mathbf{x}) = 0.75 - \prod_{i=1}^{n} x_i \leq 0$$

$$g_2(\mathbf{x}) = \sum_{i=1}^{n} x_i - 7.5n \leq 0$$

where $n = 20$ and $0 \leq x_i \leq 10$ $(i = 1, \ldots, n)$. The global maximum is unknown, the best we found is $f_2(\mathbf{x}^*) = 0.803619$ (which, to the best of our knowledge, is better than any reported value), constraint g_1 is close to being active ($g_1 = -10^{-8}$).

Maximize (Michalewicz et al., 1996):

$$f_3(\mathbf{x}) = (\sqrt{n})^n \prod_{i=1}^{n} x_i$$

$$h_1(\mathbf{x}) = \sum_{i=1}^{n} x_i^2 - 1 = 0$$

where $n = 10$ and $0 \leq x_i \leq 1$ $(i = 1, \ldots, n)$. The global maximum is at $x_i^* = 1/\sqrt{n}$ $(i = 1, \ldots, n)$ where $f_3(\mathbf{x}^*) = 1$.

Minimize (Himmelblau, 1972):

$$f_4(\mathbf{x}) = 5.3578547x_3^2 + 0.8356891x_1x_5 + 37.293239x_1 - 40792.141$$

subject to:

$$\begin{aligned}
g_1(\mathbf{x}) &= 85.334407 + 0.0056858x_2x_5 + 0.0006262x_1x_4 - 0.0022053x_3x_5 - 92 \leq 0 \\
g_2(\mathbf{x}) &= -85.334407 - 0.0056858x_2x_5 - 0.0006262x_1x_4 + 0.0022053x_3x_5 \leq 0 \\
g_3(\mathbf{x}) &= 80.51249 + 0.0071317x_2x_5 + 0.0029955x_1x_2 + 0.0021813x_3^2 - 110 \leq 0 \\
g_4(\mathbf{x}) &= -80.51249 - 0.0071317x_2x_5 - 0.0029955x_1x_2 - 0.0021813x_3^2 + 90 \leq 0 \\
g_5(\mathbf{x}) &= 9.300961 + 0.0047026x_3x_5 + 0.0012547x_1x_3 + 0.0019085x_3x_4 - 25 \leq 0 \\
g_6(\mathbf{x}) &= -9.300961 - 0.0047026x_3x_5 - 0.0012547x_1x_3 - 0.0019085x_3x_4 + 20 \leq 0
\end{aligned}$$

where $78 \leq x_1 \leq 102$, $33 \leq x_2 \leq 45$ and $27 \leq x_i \leq 45$ $(i = 3, 4, 5)$. The optimum solution is $\mathbf{x}^* = (78, 33, 29.995256025682, 45, 36.775812905788)$ where $f_4(\mathbf{x}^*) = -30665.539$. Two constraints are active (g_1 and g_6).

Minimize (Hock and Schittkowski, 1981):

$$f_5(\mathbf{x}) = 3x_1 + 0.000001x_1^3 + 2x_2 + (0.000002/3)x_2^3$$

subject to:

$$\begin{aligned}
g_1(\mathbf{x}) &= -x_4 + x_3 - 0.55 \leq 0 \\
g_2(\mathbf{x}) &= -x_3 + x_4 - 0.55 \leq 0 \\
h_3(\mathbf{x}) &= 1000\sin(-x_3 - 0.25) + 1000\sin(-x_4 - 0.25) + 894.8 - x_1 = 0 \\
h_4(\mathbf{x}) &= 1000\sin(x_3 - 0.25) + 1000\sin(x_3 - x_4 - 0.25) + 894.8 - x_2 = 0 \\
h_5(\mathbf{x}) &= 1000\sin(x_4 - 0.25) + 1000\sin(x_4 - x_3 - 0.25) + 1294.8 = 0
\end{aligned}$$

where $0 \leq x_1 \leq 1200$, $0 \leq x_2 \leq 1200$, $-0.55 \leq x_3 \leq 0.55$ and $-0.55 \leq x_4 \leq 0.55$. The best known solution (Koziel and Michalewicz, 1999) $\mathbf{x}^* = (679.9453, 1026.067, 0.1188764, -0.3962336)$ where $f_5(\mathbf{x}^*) = 5126.4981$.

Minimize (Floundas and Pardalos, 1987):

$$f_6(\mathbf{x}) = (x_1 - 10)^3 + (x_2 - 20)^3$$

subject to:

$$\begin{aligned}
g_1(\mathbf{x}) &= -(x_1 - 5)^2 - (x_2 - 5)^2 + 100 \leq 0 \\
g_2(\mathbf{x}) &= (x_1 - 6)^2 + (x_2 - 5)^2 - 82.81 \leq 0
\end{aligned}$$

where $13 \leq x_1 \leq 100$ and $0 \leq x_2 \leq 100$. The optimum solution is $\mathbf{x}^* = (14.095, 0.8429)$ where $f_6(\mathbf{x}^*) = -6961.81388$. Both constraints are active.

Minimize (Hock and Schittkowski, 1981):

$$f_7(\mathbf{x}) = x_1^2 + x_2^2 + x_1x_2 - 14x_1 - 16x_2 + (x_3 - 10)^2 + 4(x_4 - 5)^2 + (x_5 - 3)^2 + 2(x_6 - 1)^2 + 5x_7^2 + 7(x_8 - 11)^2 + 2(x_9 - 10)^2 + (x_{10} - 7)^2 + 45$$

subject to:

$$\begin{aligned} g_1(\mathbf{x}) &= -105 + 4x_1 + 5x_2 - 3x_7 + 9x_8 \leq 0 \\ g_2(\mathbf{x}) &= 10x_1 - 8x_2 - 17x_7 + 2x_8 \leq 0 \\ g_3(\mathbf{x}) &= -8x_1 + 2x_2 + 5x_9 - 2x_{10} - 12 \leq 0 \\ g_4(\mathbf{x}) &= 3(x_1 - 2)^2 + 4(x_2 - 3)^2 + 2x_3^2 - 7x_4 - 120 \leq 0 \\ g_5(\mathbf{x}) &= 5x_1^2 + 8x_2 + (x_3 - 6)^2 - 2x_4 - 40 \leq 0 \\ g_6(\mathbf{x}) &= x_1^2 + 2(x_2 - 2)^2 - 2x_1x_2 + 14x_5 - 6x_6 \leq 0 \\ g_7(\mathbf{x}) &= 0.5(x_1 - 8)^2 + 2(x_2 - 4)^2 + 3x_5^2 - x_6 - 30 \leq 0 \\ g_8(\mathbf{x}) &= -3x_1 + 6x_2 + 12(x_9 - 8)^2 - 7x_{10} \leq 0 \end{aligned}$$

where $-10 \leq x_i \leq 10$ $(i = 1, \ldots, 10)$. The optimum solution is $\mathbf{x}^* = (2.171996, 2.363683, 8.773926, 5.095984, 0.9906548, 1.430574, 1.321644, 9.828726, 8.280092, 8.375927)$ where $f_7(\mathbf{x}^*) = 24.3062091$. Six constraints are active (g_1, g_2, g_3, g_4, g_5 and g_6).

Maximize (Koziel and Michalewicz, 1999):

$$f_8(\mathbf{x}) = \frac{\sin^3(2\pi x_1)\sin(2\pi x_2)}{x_1^3(x_1 + x_2)}$$

subject to:

$$\begin{aligned} g_1(\mathbf{x}) &= x_1^2 - x_2 + 1 \leq 0 \\ g_2(\mathbf{x}) &= 1 - x_1 + (x_2 - 4)^2 \leq 0 \end{aligned}$$

where $0 \leq x_1 \leq 10$ and $0 \leq x_2 \leq 10$. The optimum is located at $\mathbf{x}^* = (1.2279713, 4.2453733)$ where $f_8(\mathbf{x}^*) = 0.095825$. The solution lies within the feasible region.

Minimize (Hock and Schittkowski, 1981):

$$f_9(\mathbf{x}) = (x_1 - 10)^2 + 5(x_2 - 12)^2 + x_3^4 + 3(x_4 - 11)^2 + 10x_5^6 + 7x_6^2 + x_7^4 - 4x_6x_7 - 10x_6 - 8x_7$$

subject to:

$$\begin{aligned} g_1(\mathbf{x}) &= -127 + 2x_1^2 + 3x_2^4 + x_3 + 4x_4^2 + 5x_5 \leq 0 \\ g_2(\mathbf{x}) &= -282 + 7x_1 + 3x_2 + 10x_3^2 + x_4 - x_5 \leq 0 \\ g_3(\mathbf{x}) &= -196 + 23x_1 + x_2^2 + 6x_6^2 - 8x_7 \leq 0 \\ g_4(\mathbf{x}) &= 4x_1^2 + x_2^2 - 3x_1x_2 + 2x_3^2 + 5x_6 - 11x_7 \leq 0 \end{aligned}$$

where $-10 \leq x_i \leq 10$ for $(i = 1, \ldots, 7)$. The optimum solution is $\mathbf{x}^* = (2.330499, 1.951372, -0.4775414, 4.365726, -0.6244870, 1.038131, 1.594227)$ where $f_9(\mathbf{x}^*) = 680.6300573$. Two constraints are active (g_1 and g_4).

Minimize (Hock and Schittkowski, 1981):

$$f_{10}(\mathbf{x}) = x_1 + x_2 + x_3$$

subject to:

$$\begin{aligned} g_1(\mathbf{x}) &= -1 + 0.0025(x_4 + x_6) \leq 0 \\ g_2(\mathbf{x}) &= -1 + 0.0025(x_5 + x_7 - x_4) \leq 0 \\ g_3(\mathbf{x}) &= -1 + 0.01(x_8 - x_5) \leq 0 \\ g_4(\mathbf{x}) &= -x_1 x_6 + 833.33252 x_4 + 100 x_1 - 83333.333 \leq 0 \\ g_5(\mathbf{x}) &= -x_2 x_7 + 1250 x_5 + x_2 x_4 - 1250 x_4 \leq 0 \\ g_6(\mathbf{x}) &= -x_3 x_8 + 1250000 + x_3 x_5 - 2500 x_5 \leq 0 \end{aligned}$$

where $100 \leq x_1 \leq 10000$, $1000 \leq x_i \leq 10000$ $(i = 2, 3)$ and $10 \leq x_i \leq 1000$ $(i = 4, \ldots, 8)$. The optimum solution is $\mathbf{x}^* = (579.3167, 1359.943, 5110.071, 182.0174, 295.5985, 217.9799, 286.4162, 395.5979)$ where $f_{10}(\mathbf{x}^*) = 7049.3307$. Three constraints are active (g_1, g_2 and g_3).

Minimize (Koziel and Michalewicz, 1999):

$$f_{11}(\mathbf{x}) = x_1^2 + (x_2 - 1)^2$$

subject to:

$$h(\mathbf{x}) = x_2 - x_1^2 = 0$$

where $-1 \leq x_1 \leq 1$ and $-1 \leq x_2 \leq 1$. The optimum solution is $\mathbf{x}^* = (\pm 1/\sqrt{2}, 1/2)$ where $f_{11}(\mathbf{x}^*) = 0.75$.

Maximize (Koziel and Michalewicz, 1999):

$$f_{12}(\mathbf{x}) = (100 - (x_1 - 5)^2 - (x_2 - 5)^2 - (x_3 - 5)^2)/100$$

subject to:

$$g(\mathbf{x}) = (x_1 - p)^2 + (x_2 - q)^2 + (x_3 - r)^2 - 0.0625 \leq 0$$

where $0 \leq x_i \leq 10$ $(i = 1, 2, 3)$ and $p, q, r = 1, 2, \ldots, 9$. The feasible region of the search space consists of 9^3 disjointed spheres. A point (x_1, x_2, x_3) is feasible if and only if there exist p, q, r such that the above inequality holds. The optimum is located at $\mathbf{x}^* = (5, 5, 5)$ where $f_{12}(\mathbf{x}^*) = 1$. The solution lies within the feasible region.

Minimize (Hock and Schittkowski, 1981):

$$f_{13}(\mathbf{x}) = e^{x_1 x_2 x_3 x_4 x_5}$$

subject to:

$$\begin{aligned} h_1(\mathbf{x}) &= x_1^2 + x_2^2 + x_3^2 + x_4^2 + x_5^2 - 10 = 0 \\ h_2(\mathbf{x}) &= x_2 x_3 - 5 x_4 x_5 = 0 \\ h_3(\mathbf{x}) &= x_1^3 + x_2^3 + 1 = 0 \end{aligned}$$

where $-2.3 \leq x_i \leq 2.3$ $(i = 1, 2)$ and $-3.2 \leq x_i \leq 3.2$ $(i = 3, 4, 5)$. The optimum solution is $\mathbf{x}^* = (-1.717143,\ 1.595709,\ 1.827247,\ -0.7636413,\ -0.763645)$ where $f_{13}(\mathbf{x}^*) = 0.0539498$.

This page intentionally left blank

III

MULTI-OBJECTIVE OPTIMIZATION

This page intentionally left blank

Chapter 5

EVOLUTIONARY MULTI-OBJECTIVE OPTIMIZATION: A CRITICAL REVIEW

Carlos A. Coello Coello

"We will say that members of a collectivity enjoy maximum ophelimity in a certain position when it is impossible to find a way of moving from that position very slightly in such a manner that the ophelimity enjoyed by each of the individuals in the collectivity increases or decreases. That is to say, any small displacement in departing from that position necessarily has the effect of increasing the ophelimity which certain individuals enjoy, and decreasing that which others enjoy, of being agreeable to some and disagreeable to others"

—Vilfredo Pareto, *Manual of Political Economy,* 1896

Abstract In this chapter, we will review some of the most representative research in the field of evolutionary multiobjective optimization. We will discuss the historical roots of multiobjective optimization, the motivation to use evolutionary algorithms, and the most popular techniques currently in use. Then, we will discuss some of the research currently under way, including our own. At the end, we will provide what we consider to be some of the most promising paths of future research.

Keywords: evolutionary multiobjective optimization, evolutionary algorithms, vector optimization, multiobjective optimization, genetic algorithms, multicriteria optimization

1. Introduction

Most optimization problems naturally have several objectives to be achieved (normally conflicting with each other), but in order to simplify their solution, they are treated as if they had only one (the remaining objectives are normally handled as constraints). These problems with several objectives, are called "multiobjective" or "vector" optimization problems, and were originally studied in the context of economics. However, scientists and engineers soon realized that such problems naturally arise in all areas of knowledge.

Over the years, the work of a considerable amount of operational researchers has produced an important number of techniques to deal with multiobjective optimization problems (Miettinen, 1998). However, it was until relatively recently that researchers realized of the potential of evolutionary algorithms (EAs) in this area.

This chapter will review the most important research in the area now called Evolutionary Multi-Objective Optimization, or EMOO for short. The importance of this field is reflected by a significant increment (mainly during the last five years) of technical papers in international conferences and peer-reviewed journals, special sessions in international conferences and interest groups in the Internet[1].

The organization of this chapter is the following: first, we will provide some basic concepts used in multiobjective optimization. Then, we will briefly discuss the historical roots of this discipline, and the motivation for using evolutionary algorithms. After that, we will do a critical review of the most popular EMOO techniques currently available, including some of their applications. Finally, we will discuss some of the research currently under way, including our own. We will finish this chapter with a brief discussion of what we consider to be some of the most promising paths of future research.

2. Definitions

We are interested in solving multiobjective optimization problems (MOPs) of the form:

$$\text{minimize } [f_1(\mathbf{x}), f_2(\mathbf{x}), \ldots, f_k(\mathbf{x})] \tag{5.1}$$

subject to the m inequality constraints:

[1]The author maintains an EMOO repository which currently includes over 650 bibliographical entries at: http://www.lania.mx/~ccoello/EMOO/ with a mirror at http://www.jeo.org/emo/

$$g_i(\mathbf{x}) \geq 0 \quad i = 1, 2, \ldots, m \tag{5.2}$$

and the p equality constraints:

$$h_i(\mathbf{x}) = 0 \quad i = 1, 2, \ldots, p \tag{5.3}$$

where k is the number of objective functions $f_i : \mathbb{R}^n \rightarrow \mathbb{R}$. We call $\mathbf{x} = [x_1, x_2, \ldots, x_n]^T$ the vector of decision variables. We wish to determine from among the set $\mathcal{F}$ of all numbers which satisfy (5.2) and (5.3) the particular set $x_1^*, x_2^*, \ldots, x_n^*$ which yields the optimum values of all the objective functions.

2.1 Pareto Optimum

It is rarely the case that there is a single point that simultaneously optimizes all the objective functions. Therefore, we normally look for "trade-offs", rather than single solutions when dealing with multiobjective optimization problems. The notion of "optimum" is therefore, different. The most commonly adopted notion of optimality is that originally proposed by Francis Ysidro Edgeworth (1881) and later generalized by Vilfredo Pareto (1896). Although some authors call *Edgeworth-Pareto optimum* to this notion (see for example Stadler (1988)), we will use the most commonly accepted term: *Pareto optimum.*

We say that a vector of decision variables $\mathbf{x}^* \in \mathcal{F}$ is *Pareto optimal* if there does not exist another $\mathbf{x} \in \mathcal{F}$ such that $f_i(\mathbf{x}) \leq f_i(\mathbf{x}^*)$ for all $i = 1, \ldots, k$ and $f_j(\mathbf{x}) < f_j(\mathbf{x}^*)$ for at least one j.

In words, this definition says that $\mathbf{x}^*$ is Pareto optimal if there exists no feasible vector of decision variables $\mathbf{x} \in \mathcal{F}$ which would decrease some criterion without causing a simultaneous increase in at least one other criterion. Unfortunately, this concept almost always gives not a single solution, but rather a set of solutions called the *Pareto optimal set.* The vectors $\mathbf{x}^*$ correspoding to the solutions included in the Pareto optimal set are called *nondominated.* The plot of the objective functions whose nondominated vectors are in the Pareto optimal set is called the *Pareto front.*

3. Historical Roots

John von Neumann and Oskar Morgenstern (1944) were the first to recognize the existence of optimization problems in economics that were "a peculiar and disconcerting mixture of several conflicting problems". However, no real contribution to the solution of such problems was made until the 1950s.

Harold W. Kuhn and Albert W. Tucker (1951) introduced a vector-valued objective function in mathematical programming—a *vector maximum problem,* and derived the optimality conditions for efficient solutions. The so-called "proper efficiency" in the context of multiobjective optimization was also formulated in this seminal paper that many consider as the first serious attempt to derive a theory in this area. This same direction was later followed by Arrow et al. (1953) who used the term "admissible" instead of "efficient" points.

However, multiobjective optimization theory remained relatively undeveloped during the 1950s, and the subject was scarcely covered by only a few authors (see for example (Koopman, 1953; Karlin, 1959)).

The application of multiobjective optimization to domains outside economics began with the work of Tjalling Koopmans (1951) in production theory and with the work of Marglin (1967) in water resources planning. The first application of multiobjective optimization in engineering was in the early 1960s (Zadeh, 1963), but its use became generalized until the 1970s (Stadler, 1975; Cohon, 1978).

Good reviews of existing mathematical programming techniques for multiobjective optimization can be found in a variety of references (see for example (Cohon and Marks, 1975; Hwang et al., 1980; Miettinen, 1998)).

Evolutionary algorithms have been successfully applied to a variety of optimization problems with very large (intractable) search spaces, noise, non-differentiable and even dynamic objective functions in the last few years (Goldberg, 1989; Michalewicz, 1996; Mitchell, 1996; Gen and Cheng, 1997).

The potential of evolutionary algorithms in this field was hinted in the late 1960s by Rosenberg (1967), but the first implementation was produced until the mid-1980s (Schaffer, 1985). Evolutionary algorithms seem particularly appropriate to solve multiobjective optimization problems because they deal simultaneously with a set of possible solutions (the so-called population) which allows us to find several members of the Pareto optimal set in a single run of the algorithm, instead of having to perform a series of separate runs as in the case of the traditional mathematical programming techniques. Additionally, evolutionary algorithms are less succeptible to the shape or continuity of the Pareto front (e.g., they can easily deal with non-convex Pareto fronts), whereas these two issues are a real concern for mathematical programming techniques.

Evolutionary algorithms are not the only heuristic technique that has been used to solve multiobjective optimization problems. The good performance exhibited by some algorithms (e.g., tabu search and simulated annealing) in combinatorial optimization problems has led researchers to

develop multiobjective versions of them (Hansen, 1996; Ehrgott, 2000; Czyzak and Jaszkiewicz, 1997; Gandibleux et al., 1997; Romero and Manzanares, 1999). Some researchers have also suggested hybrids between genetic algorithms and other heuristics (e.g., tabu search (Kurahashi and Terano, 2000)) for multiobjective optimization. Nevertheless, our review will only concentrate on evolutionary multiobjective optimization techniques.

4. A Quick Survey of EMOO Approaches

A considerable number of EMOO techniques have been proposed in the last few years and it is not our intention to enumerate them all in this chapter (interested readers should refer to (Coello, 1999) and (Veldhuizen, 1999) for more detailed surveys of EMOO approaches). Therefore, we will concentrate our discussion on those techniques that have been more popular among researchers or that are very recent (and promising according to our own personal criterion). The techniques discussed are the following: Aggregating functions, VEGA, MOGA, NSGA, NPGA, target vector approaches and two recent approaches: PAES and SPEA.

4.1 Aggregating functions

Knowing that an EA[2] needs scalar fitness information to work, it is almost natural to propose a combination of all the objectives into a single one using either an addition, multiplication or any other combination of arithmetical operations that we could devise. In fact, this is also the oldest mathematical programming method for multiobjective optimization, since it can be derived from the Kuhn-Tucker conditions for inxnondominated solutions (Kuhn and Tucker, 1951). An example of this approach is a sum of weights of the form:

$$\min \sum_{i=1}^{k} w_i f_i(\mathbf{x}) \tag{5.4}$$

where $w_i \geq 0$ are the weighting coefficients representing the relative importance of the k objective functions of our problem. It is usually assumed that

[2]We will use the generic term *Evolutionary Algorithm* throughout this chapter, although most of the EMOO approaches discussed use genetic algorithms.

$$\sum_{i=1}^{k} w_i = 1 \tag{5.5}$$

4.1.1 Strengths and weaknesses. The main strengths of this method are its simplicity and efficiency (computationally speaking). It can work properly in simple (convex) MOPs with few objective functions. This approach is normally used to generate a single (or a few) nondominated solution that can be used as an initial solution for other techniques. One of its main weaknesses is the difficulty to determine the set of weights that can appropriately scale the objectives when we do not have enough information about the problem. Its most serious drawback is that it cannot generate proper members of the Pareto optimal set when the Pareto front is concave regardless of the weights used (Das and Dennis, 1997).

4.1.2 Some Applications.

- Task planning (Jakob et al., 1992).
- System-level synthesis (Blickle et al., 1996).
- Truss optimization (Liu et al., 1998).

4.2 VEGA

David Schaffer (1985) proposed an approach that he called the *Vector Evaluated Genetic Algorithm* (VEGA), and that differed of the simple genetic algorithm (GA) only in the way in which selection was performed. This operator was modified so that at each generation a number of sub-populations was generated by performing proportional selection according to each objective function in turn. Thus, for a problem with k objectives and a population size of M, k sub-populations of size M/k each would be generated. These sub-populations would be shuffled together to obtain a new population of size M, on which the GA would apply the crossover and mutation operators in the usual way.

The solutions generated by VEGA are locally nondominated, but not necessarily globally nondominated. VEGA presents the so-called "speciation" problem (i.e., we could have the evolution of "species" within the population which excel on different objectives). This problem arises because this technique selects individuals who excel in one objective, without looking at the others. The potential danger doing that is that we could have individuals with what Schaffer (1985) called "middling"

performance[3] in all dimensions, which could be very useful for compromise solutions, but that will not survive under this selection scheme, since they are not in the extreme for any dimension of performance (i.e., they do not produce the best value for any objective function, but only moderately good values for all of them). Speciation is undesirable because it is opposed to our goal of finding compromise solutions.

4.2.1 Strengths and weaknesses. The main advantages of this technique are its simplicity and its efficiency. However, as we mentioned before, the "middling" problem prevents the technique from finding the compromise solutions that we normally aim to produce. In fact, if proportional selection is used with VEGA (as Schaffer did), the shuffling and merging of all the sub-populations corresponds to averaging the fitness components associated with each of the objectives (Richardson et al., 1989). In other words, under these conditions, VEGA behaves as an aggregating approach and therefore, it is subject to the same problems of such techniques.

4.2.2 Some Applications.

- Groundwater pollution containment (Ritzel et al., 1994).
- Constraint-handling (Surry et al., 1995; Coello, 2000c).
- Scheduling (Hilliard et al., 1989).

4.3 MOGA

Fonseca and Fleming (1993) proposed the *Multi-Objective Genetic Algorithm* (MOGA). The approach consists of a scheme in which the rank of a certain individual corresponds to the number of individuals in the current population by which it is dominated. All nondominated individuals are assigned rank 1, while dominated ones are penalized according to the population density of the corresponding region of the trade-off surface.

Fitness assignment is performed in the following way (Fonseca and Fleming, 1993):

1 Sort population according to rank.

2 Assign fitness to individuals by interpolating from the best (rank 1) to the worst $(\text{rank } n \leq N)$ in the way proposed by Goldberg (1989)

[3]By "middling", Schaffer meant an individual with acceptable performance, perhaps above average, but not outstanding for any of the objective functions.

(the so-called Pareto ranking assignment process), according to some function, usually linear, but not necessarily.

3 Average the fitnesses of individuals with the same rank, so that all of them will be sampled at the same rate.

MOGA is combined with mating restrictions and sharing on the objective function values to preserve diversity (Deb and Goldberg, 1989). The authors of this method also provided some guidelines regarding the way in which niche sizes can be estimated.

4.3.1 Strengths and weaknesses. MOGA has been a very popular EMOO technique (particularly within the control community), not only because it is relatively simple to implement, but also because of its good overall performance (Coello, 1996). Its main weakness is its dependence on the sharing factor (how to maintain diversity is the main issue when dealing with EMOO approaches in general). However, as indicated before, Fonseca and Fleming (1993) have provided some guidelines regarding the way to compute niche sizes.

4.3.2 Some Applications.

- Controllers design (Tan and Li, 1997; Chipperfield and Fleming, 1995; Schroder et al., 1997)
- Co-synthesis of hardware-software embedded systems (Dick and Jha, 1998)
- Truss design (Narayanan and Azarm, 1999)

4.4 NSGA

The *Nondominated Sorting Genetic Algorithm* (NSGA) was proposed by Srinivas and Deb (1994), and is based on several layers of classifications of the individuals. Before selection is performed, the population is ranked on the basis of domination (using Pareto ranking): all nondominated individuals are classified into one category (with a dummy fitness value, which is proportional to the population size). To maintain the diversity of the population, these classified individuals are shared (in decision variable space) with their dummy fitness values. Then this group of classified individuals is removed from the population and another layer of nondominated individuals is considered (i.e., the remainder of the population is re-classified). The process continues until all individuals in the population are classified. Since individuals in the first

front have the maximum fitness value, they always get more copies than the rest of the population.

4.4.1 Strengths and weaknesses. Some researchers have reported that NSGA has a lower overall performance than MOGA, and it seems to be also more sensitive to the value of the sharing factor than MOGA (Coello, 1996; Veldhuizen, 1999). However, Deb et al. (2000a,2000b) have recently proposed a new version of this algorithm, called NSGA-II, which is more efficient (computationally speaking), uses elitism and a crowded comparison operator that keeps diversity without specifying any additional parameters. The new approach has not been extensively tested yet, but it certainly looks promising.

4.4.2 Some Applications.

- Investment portfolio optimization (Vedarajan et al., 1997).
- Optimization of low-thrust interplanetary spacecraft trajectories (Hartmann et al., 1998).
- Optimization of an industrial nylon 6 semibatch reactor (Mitra et al., 1998).

4.5 NPGA

Horn et al. (1994) proposed the *Niched Pareto Genetic Algorithm* which uses a tournament selection scheme based on Pareto dominance. Two individuals are compared against a set of members of the population (typically, 10% of the population size). When both competitors are either dominated or nondominated (i.e., when there is a tie), the result of the tournament is decided through fitness sharing in the objective domain (a technique called *equivalent class sharing* was used in this case) (Horn et al., 1994).

4.5.1 Strengths and weaknesses. Since this approach does not apply Pareto ranking to the entire population, but only to a segment of it at each run, its main strength are that it is faster than MOGA and NSGA[4]. Furthermore, it also produces good nondominated fronts that can be kept for a large number of generations (Coello, 1996). However, its main weakness is that besides requiring a sharing factor, this

[4]Pareto ranking is $O(kM^2)$, where k is the number of objectives and M is the population size

approach also requires an additional parameter: the size of the tournament.

4.5.2 Some Applications.

- Design of laminated ceramic composites (Belegundu et al., 1994).
- Airfoil design (Quagliarella and Vicini, 1997).
- Manufacturing cell formation problems (Pierreval and Plaquin, 1998).

4.6 Target Vector Approaches

Under this name we will consider approaches in which the decision maker has to assign targets or goals that wishes to achieve for each objective[5]. The EA in this case, tries to minimize the difference between the current solution found and the vector of goals (different metrics can be used for that purpose). The most popular techniques included here are hybrids with: Goal Programming (Deb, 1999c; Wienke et al., 1992), Goal Attainment (Wilson and Macleod, 1993; Zebulum et al., 1998) and the min-max approach (Hajela and Lin, 1992; Coello and Christiansen, 1998).

4.6.1 Strengths and weaknesses. The main strength of these methods is their efficiency (computationally speaking) because they do not require a Pareto ranking procedure. However, their main weakness is the definition of the desired goals which requires some extra computational effort. Furthermore, these techniques will yield a nondominated solution only if the goals are chosen in the feasible domain, and such condition may certainly limit their applicability.

4.6.2 Some Applications.

- Design of multiplierless IIR filters (Wilson and Macleod, 1993).
- Structural optimization (Sandgren, 1994; Hajela and Lin, 1992).
- Optimization of the counterweight balancing of a robot arm (Coello et al., 1998).

[5] Although target vector approaches can be considered as another aggregating approach, we decided to discuss them separately because these techniques can generate (under certain conditions) non-convex portions of the Pareto front, whereas approaches based on weighted sums cannot.

4.7 Recent approaches

Recently, several new EMOO approaches have been developed. We consider important to discuss briefly at least two of them: PAES and SPEA.

The *Pareto Archived Evolution Strategy* (PAES) was introduced by Knowles and Corne (2000a). This approach is very simple: it uses a (1+1) evolution strategy (i.e., a single parent that generates a single offspring) together with a historical archive that records all the nondominated solutions previously found (such archive is used as a comparison set in a way analogous to the tournament competitors in the NPGA). PAES also uses a novel approach to keep diversity, which consists of a crowding procedure that divides objective space in a recursive manner. Each solution is placed in a certain grid location based on the values of its objectives. A map of such grid is maintained, indicating the amount of solutions that reside in each grid location. Since the procedure is adaptive, no extra parameters are required (except for the number of divisions of the objective space). Furthermore, the procedure has a lower computational complexity than traditional niching methods. PAES has been used to solve the off-line routing problem (Knowles and Corne, 1999) and the adaptive distributed database management problem (Knowles and Corne, 2000).

The *Strength Pareto Evolutionary Algorithm* (SPEA) was introduced by Zitzler and Thiele (1999). This approach was conceived as a way of integrating different EMOO techniques. SPEA uses an archive containing nondominated solutions previously found (the so-called external nondominated set). At each generation, nondominated individuals are copied to the external nondominated set. For each individual in this external set, a strength value is computed. This strength is similar to the ranking value of MOGA, since it is proportional to the number of solutions to which a certain individual dominates. The fitness of each member of the current population is computed according to the strengths of all external nondominated solutions that dominate it. Additionally, a clustering technique is used to keep diversity. SPEA has been used to explore trade-offs of software implementations for DSP algorithms (Zitzler et al., 1999) and to solve 0/1 knapsack problems (Zitzler and Thiele, 1999).

Recently, we have been experimenting with a micro-GA (a GA with small population and a reinitialization mechanism (Krishnakumar, 1989)) for multiobjective optimization (Coello and Toscano, 2001). Our approach uses two memories: 1) a population memory, which is used as the source of diversity, and 2) an external memory, which is used to

archive members of the Pareto optimal set found during the evolutionary process. Our micro-GA uses a population of four individuals, which undergo binary tournament selection, two point crossover and uniform mutation until nominal convergence is achieved (a small number of iterations is used in our case, but other criteria could also be used). Through the use of different forms of elitism and a reinitialization process that mixes good solutions previously found with random solutions, we gradually approach the true Pareto front of a problem. To keep diversity, we use an approach similar to the adaptive grid proposed by Knowles and Corne (2000a). The idea is that once the archive that stores nondominated vectors (i.e., the external memory) has reached its limit, we divide the search space that this archive covers, assigning a set of coordinates to each vector. Then, each newly generated nondominated vector will be accepted only if the geographical location to where it belongs is less populated than the most crowded location, or if it belongs to a location outside the previously specified boundaries (i.e., if it forms a new niche). The approach has a very low computational cost (with respecto to Pareto ranking) and we can regulate the amount of points from the Pareto front that we wish to find through the size of the external memory. Our preliminary results indicate that our micro-GA is able to generate the Pareto front of difficult test functions (i.e., disconnected and concave Pareto fronts) that have been previously adopted to evaluate EMOO techniques (Coello and Toscano, 2001).

5. Current Research

Being a very active area of research, EMOO has seen a lot of changes in the last few years and the research trends are constantly changing. We will focus our discussion in this section to two main areas that currently interest us: constraint-handling for evolutionary optimization, and incorporation of preferences into an EMOO algorithm. These two areas have not been studied in enough depth and, from our particular point of view, seem very promising.

5.1 Handling Constraints

An interesting application of EMOO techniques that we have recently explored is in constraint-handling (for single-objective evolutionary optimization). The most straightforward approach is to redefine the single-objective optimization of $f(\mathbf{x})$ as a multiobjective optimization problem in which we will have $m + 1$ objectives, where m is the number of con-

straints[6]. Then, we can apply any EMOO technique to the new vector $\bar{v} = (f(\mathbf{x}), f_1(\mathbf{x}), \ldots, f_m(\mathbf{x}))$, where $f_1(\mathbf{x}), \ldots, f_m(\mathbf{x})$ are the original constraints of the problem. An ideal solution $\mathbf{x}$ would thus have $f_i(\mathbf{x})=0$ for $1 \leq i \leq m$ and $f(\mathbf{x}) \leq f(\mathbf{y})$ for all feasible $\mathbf{y}$ (assuming minimization).

However, it should be clear that in single-objective optimization problems we do not want just good trade-offs; we want to find the best possible solutions that do not violate any constraints. Therefore, a mechanism such as Pareto ranking may be useful to approach the feasible region, but once we arrive to it, we will need to guide the search with a different mechanism so that we can reach the global optimum. In order to achieve this goal, we should also be able to maintain diversity in the population. These aspects are the main focus of the research briefly reviewed in this section.

Surry et al. (1997) proposed the use of Pareto ranking and VEGA to handle constraints. In their approach, called COMOGA, the population is ranked based on constraint violations (counting the number of individuals dominated by each solution). Then, one portion of the population is selected based on constraint ranking, and the rest based on real cost (fitness) of the individuals. COMOGA compared fairly with a penalty-based approach in a pipe-sizing problem, and was less sensitive to changes in the parameters, but the results achieved were not better than those found with a penalty function (Surry and Radcliffe, 1997). It should be added that COMOGA requires several extra parameters, although its authors argue that the technique is not particularly sensitive to the values of such parameters.

Parmee and Purchase (1994) implemented a version of VEGA that handled the constraints of a gas turbine problem as objectives to allow a GA to locate a feasible region within the highly constrained search space of this application. However, VEGA was not used to further explore the feasible region, and instead the authors used specialized operators that would create a variable-size hypercube around each feasible point to help the GA to remain within the feasible region at all times. It is important to notice that no real attempt to reach the global optimum was made in this case.

Camponogara and Talukdar (1997) proposed to restate a single objective optimization problem in such a way that two objectives would be considered: the first would be to optimize the original objective function and the second would be to minimize:

[6]The assumption that we have m constraints will hold throughout this section.

$$\Phi(\mathbf{x}) = \sum_{i=1}^{m} \max[0, g_i(\mathbf{x})]^{\beta} \tag{5.6}$$

where β is normally 1 or 2.

Once the problem is redefined, nondominated solutions with respect to the two new objectives are generated. The solutions found define a search direction $d = (x_i - x_j)/|x_i - xj|$, where $x_i \in S_i$, $x_j \in S_j$, and S_i and S_j are Pareto sets. The direction search d is intended to simultaneously minimize all the objectives. Line search is performed in this direction so that a solution x can be found such that x dominates x_i and x_j (i.e., x is a better compromise than the two previous solutions found). Line search takes the place of crossover in this approach, and mutation is essentially the same, where the direction d is projected onto the axis of one variable j in the solution space. Additionally, a process of eliminating half of the population is applied at regular intervals (only the less fitted solutions are replaced by randomly generated points).

This approach has obvious problems to keep diversity, as it is reflected by the need to discard the worst individuals at each generation. Also, the use of line search increases the computational cost of the approach and it is not clear what is the impact of the segment chosen to search in the overall performance of the algorithm.

Jiménez et al. (1999) proposed the use of a min-max approach (Chankong and Haimes, 1983) to handle constraints. The main idea of this technique is to apply a set of simple rules to decide the (binary tournament) selection process:

1 If the two individuals being compared are both feasible, then select based on the minimum value of the objective function.

2 If one of the two individuals being compared is feasible and the other one is infeasible, then select the feasible individual.

3 If both individuals are infeasible, then select based on the maximum constraint violation ($\max\ g_j(\mathbf{x})$, for $j = 1, \ldots, m$). The individual with the lowest maximum violation wins.

A subtle problem with this approach is that the evolutionary process first concentrates only on the constraint satisfaction problem and therefore it samples points in the feasible region essentially at random (Surry et al., 1995). This means that in some cases (e.g., when the feasible region is disjoint) we might land in an inappropriate part of the feasible region from which we will not be able to escape. However, this approach may be a good alternative to find a feasible point in a heavily constrained search space. Deb (2000) proposed a similar approach but

using tournament selection based on feasibility. However, niching was required to maintain diversity in the population.

Coello (2000c) proposed the use of a population-based multiobjective optimization technique such as VEGA to handle each of the constraints of a single-objective optimization problem as an objective. At each generation, the population is split into $m + 1$ sub-populations (m is the number of constraints), so that a fraction of the population is selected using the (unconstrained) objective function as its fitness and another fraction uses the first constraint as its fitness and so on. This approach provided good results in several optimization problems (Coello, 2000c). Its main disadvantage was related to scalability issues. However, in a recent application in combinational circuit design we were able to successfully deal with up to 49 objective functions (Coello et al., 2000b). Furthermore, the approach showed an important improvement (in terms of efficiency) with respect to a previous GA-based approach developed by us for the same task (Coello et al., 2000a).

Recently, we have also explored the use of selection based on dominance (which was defined in terms of feasibility) to handle constraints (Coello, 2000a). Our approach uses stochastic universal sampling so that the selection pressure is not too high and no extra procedures are required to maintain diversity. Also, adaptive crossover and mutation rates were adopted as part of the approach.

The key for future research in this area is not only to adapt other EMOO approaches to handle constraints, but to exploit domain knowledge as much as possible. An example of this is the recent work by Ray et al. (2000) in which solutions are ranked separately based on the value of their objective functions and their constraints. Then a set of mating restrictions are applied based on the information that each individual has of its own feasibility (this idea was inspired on an earlier approach by Hinterding and Michalewicz (1998)), so that the global optimum can be reached through cooperative learning.

Other approaches are also possible. For example, we could combine an EMOO approach with a mechanism to incorporate preferences from the user (the topic discussed in the next section). Such preferences, however, could be directly derived from the problem (using the domain knowledge available), instead of requiring an active participation from the user.

5.2 Incorporation of Preferences

By looking at most of the EMOO papers in the literature, one gets the impression that researchers seem to forget that the solution of a

MOP really involves three stages: measurement, search, and decision making. Most EMOO research tends to concentrate on issues related to the search of nondominated vectors. However, these nondominated vectors do not provide any insight into the process of decision making itself (the decision maker (DM) still has to choose manually one of the several alternatives produced), since they are really a useful generalization of a utility function under the conditions of minimum information (i.e., all attributes are considered as having equal importance; in other words, the DM does not express any preferences of the attributes). Thus, the issue is how to incorporate the DM's preferences into an EMOO approach as to guide the search only to the regions of main interest for the DM.

One way to classify techniques that incorporate preferences from the DM is based on the moment (within the search process) at which preferences are expressed. According to this criterion, preferences can be expressed (Horn, 1997): *a priori, a posteriori,* or in an *interactive* way when using EAs.

If preferences are expressed *a priori,* the DM has to define them in advance (before actually performing the search). An example of this are the aggregating approaches discussed in Section 4.1. Shaw and Fleming (1997), Greenwood et al. (1997), and Cvetković and Parmee (2000) have proposed *a priori* schemes to incorporate preferences into an EMOO approach.

In the second case, we search first, and decide later. Most EMOO approaches (those that use Pareto ranking) fall into this category. In this case, we use an EA to search the "best possible" alternatives, where "best possible" normally means members of the Pareto optimal set. Massebeuf et al. (1999) proposed an *a posteriori* scheme to incorporate preferences into an EMOO approach.

The third case is the less common in the EA literature: approaches that allow to guide the search of the EA using preferences from the DM, but in an interactive way (i.e., assuming that such preferences can change over time). Tanino et al. (1993) and Fonseca and Fleming (1993,1998) proposed *interactive* schemes to incorporate preferences into an EMOO approach[7].

The Operations Research (OR) literature has normally favored *interactive* approaches for several reasons (Monarchi et al., 1973):

1 It is normally the case that the DM wishes to find trade-offs that satisfy only a certain set of criteria, instead of wishing to find

[7]The approach was also used to handle constraints.

solutions that are the best trade-off considering all criteria at the same time.

2 The preferences of the DM can (and normally do) change over time.

3 The DM normally learns through the search process and tends to change (in consequence) his aspirations or desires.

If we analyze the literature on multi-criteria decision making (MCDM), we find another way of classifying approaches to incorporate preferences. In this case, two main lines of thought are normally considered: the so-called French School, which is based on the outranking concept (Vincke, 1995) and the American School, which is based on the Multi-Attribute Utility Theory (MAUT) (Hwang and Masud, 1979). Both outranking and MAUT can be used a priori, a posteriori or in an interactive way.

Outranking relationships are built under the form of pairwise comparisons of the objects under study (a graph representing preferences is normally used). Pairs of objects are compared to determine if there exists preference, indifference, or incomparability between them. Weights for each objective are derived from these pairwise comparisons. It is important, however, to be aware of the fact that these pairwise comparisons may lead to intransitive or incomplete relations (van Huylenbroeck, 1995). The main drawbacks of outranking approaches are their high computational cost when there is a large amount of alternatives, the high amount of parameters that they require, and the difficulties to define some of these alternatives (e.g., the "degree of credibility") (Brans et al., 1986). Rekiek et al. (2000) and Massebeuf et al. (1999) have proposed EMOO approaches that incorporate preferences using PROMETHEE (Brans et al., 1986) (Preference Ranking Organization Method for Enrichment Evaluations), which is an outranking approach.

MAUT is based on the formulation of an overall utility function. Although it is normally assumed that such utility function can be obtained, when that is not possible, then nondominated solutions can be used (i.e., we assume that all objectives are given the same importance). Certain flexibility can be obtained through the concept of "weak dominance" (Loucks, 1975), which can be used to express a certain lack of conviction. It is also possible for the DM to express indifference, which means that both vectors under comparison are equivalent and that it does not matter which one is selected. It is worth mentioning that "indifference" is not the same that "incomparability" (as defined in outranking methods), because the second indicates vectors with strong opposite merits (van Huylenbroeck, 1995). MAUT does not allow intransitivities to occur.

This considerably simplifies the modelling of the preferences. However, it is not very difficult to produce an example in which intransitivities "naturally" emerge (see for example (van Huylenbroeck, 1995)). Greenwood et al. (1997) and Cvetković and Parmee (2000) have proposed EMOO approaches that incorporate preferences using utility functions.

Despite this research, there is an obvious lack of models for the incorporation of preferences into an EMOO approach. Issues such as scalability and the presence of several DMs deserve special attention when devising such a model.

As we have indicated before (Coello, 2000b), we believe that there are several approaches from OR that could be easily coupled with EAs. Approaches such as PROTRADE (PRObabilistic TRAde-off DEvelopment method) (Goicoechea et al., 1979) and SEMOPS (Sequential Multi-Objective Problem Solving method) (Monarchi et al., 1973) could be easily tailored to incorporate preferences into EMOO approaches. Both approaches are interactive and assume a degree of uncertainty from the DM with respect to the trade-offs of the objectives under study. Compromise programming (Duckstein, 1984) is also promising, and it has in fact been used by some EMOO researchers (see for example (Deb, 1999a; Bentley and Wakefield, 1997)). However, more complex articulations of preferences are possible if the approach is used interactively (it has been normally used as an *a priori* technique).

We believe that a key issue to foster the development of this area in the future is that EMOO researchers be aware of the work done by operational researchers in MCDM. It should be clear to EMOO researchers that searching efficiently nondominated vectors is not the only important topic in multiobjective optimization.

6. Future Research Paths

As has been indicated before in some of the sections of this chapter, a lot of work remains to be done in this area. We will describe next some of the future research paths that we consider most promising in this area:

- Combinatorial Optimization: We believe that EMOO researchers can benefit from the considerable amount of work done in combinatorial optimization by relying on multiobjective versions of such problems. Such problems are not only challenging, but have also been studied in great depth (Ehrgott and Gandibleux, 2000). The benchmarks available for problems like the 0/1 knapsack can be used to test EMOO approaches. Such idea has been explored by a few EMOO researchers (for example (Zitzler and Thiele,

1999; Jaszkiewicz, 2000)), but more work in this direction is still necessary.

- Efficient data structures: EMOO researchers have paid little attention to the data structures used to store nondominated vectors. Operational researchers have used, for example, domination-free quad trees where a nondominated vector can be retrieved from the tree very efficiently. Checking if a new vector is dominated by the vectors in one of these trees can also be done very efficiently (Habenicht, 1982). Efficiency has been emphasized in EMOO research until recently (Deb et al., 2000a), mainly regarding the number of comparisons performed for ranking the population and to maintain diversity, but a lot of work is still necessary.

- Theoretical issues: There are very few theoretical studies related to EMOO, and most of them concentrate on convergence issues (Rudolph, 1998; Rudolph and Agapie, 2000; Hanne, 2000; Veldhuizen and Lamont, 1998), or on ways to compute niche sizes (Fonseca and Fleming, 1993; Horn et al., 1994). However, many other important areas have not been studied. It would be very interesting to study, for example, the structure of fitness landscapes in MOPs. Such study could provide some insights regarding the sort of problems that are particularly difficult for EAs and could also provide clues regarding the design of more powerful EMOO techniques. Also, there is a need for detailed studies of the different aspects involved in the parallelization of EMOO techniques (e.g., load balancing, impact on Pareto convergence, performance issues, etc.), including new algorithms that are more suitable for parallelization than those currently in use.

There are also several other research areas that are worth exploring. For example: development of MOP test functions (Veldhuizen and Lamont, 1999; Deb, 1999b; Deb and Meyarivan, 2000), appropriate metrics that allow us to evaluate performance in a quantitative way (Zitzler et al., 2000; Veldhuizen, 1999; Fonseca and Fleming, 1996), to study in more depth the role of local search in multiobjective optimization (Ishibuchi and Murata, 1996; Parks and Miller, 1998; Knowles and Corne, 2000; Coello and Toscano, 2001), etc. Some of these areas are actively being pursued by several researchers nowadays.

7. Summary

We have tried to give a general perspective of the research that has been done and that is currently under way in evolutionary multiobjective

optimization, including our own. Starting with a short discussion on the origins of a separate discipline devoted to the study of MOPs, we have led our discussion towards the main motivations to use EAs in these types of problems.

We have stressed the importance of studying the several issues involved in solving a MOP, rather than just focusing our research in the development of efficient procedures to generate nondominated vectors. Decision making is as important (or maybe more) than just generating trade-offs for a MOP, and most EMOO researchers seem to overlook this matter.

We have also indicated some promising research trends (from our personal perspective), from which the lack of theoretical studies remains as the area that requires more attention from EMOO researchers.

Finally, we have also surveyed the main EMOO approaches currently in use, indicating some of their applications reported in the literature, as well as their advantages and disadvantages.

But overall, one of the most reiterative issues that we have underlined in this chapter has been the importance of relying on the work done in OR as a basis for pursuing research in EMOO. The awareness of the important contributions to multiobjective optimization that operational researchers have made will help EMOO researchers to have a wider perspective of the field and a deeper understanding of the fundamental problems that need to be solved in this discipline.

Acknowledgements

The author acknowledges support from the mexican Consejo Nacional de Ciencia y Tecnología (CONACyT) through project number 34201-A.

References

Arrow, K. J., Barankin, E. W., and Blackwell, D. (1953). Admissible Points of Convex Sets. In Kuhn, H. W. and Tucker, A. W., editors, *Contributions to the Theory of Games,* pages 87–91. Princeton University Press, Princeton, New Jersey.

Belegundu, A. D., Murthy, D. V., Salagame, R. R., and Constants, E. W. (1994). Multiobjective Optimization of Laminated Ceramic Composites Using Genetic Algorithms. In *Fifth AIAA/USAF/NASA Symposium on Multidisciplinary Analysis and Optimization,* pages 1015–1022, Panama City, Florida. AIAA. Paper 84-4363-CP.

Bentley, P. J. and Wakefield, J. P. (1997). Finding Acceptable Solutions in the Pareto-Optimal Range using Multiobjective Genetic Algorithms. In Chawdhry, P. K., Roy, R., and Pant, R. K., editors, *Soft*

Computing in Engineering Design and Manufacturing, Part 5, pages 231–240, London. Springer Verlag London Limited. (Presented at the 2nd On-line World Conference on Soft Computing in Design and Manufacturing (WSC2)).

Blickle, T., Teich, J., and Thiele, L. (1996). System-level synthesis using evolutionary algorithms. Technical Report TIK Report-Nr. 16, Computer Engineering and Communication Networks Lab (TIK), Swiss Federal Institute of Technology (ETH), Gloriastrasse 35, 8092 Zurich.

Brans, J. P., Vincke, P., and Mareschal, B. (1986). How to select and how to rank projects: the PROMETHEE method. *European Journal of Operational Research,* 24(2):228–238.

Camponogara, E. and Talukdar, S. N. (1997). A Genetic Algorithm for Constrained and Multiobjective Optimization. In Alander, J. T., editor, *3rd Nordic Workshop on Genetic Algorithms and Their Applications (3NWGA),* pages 49–62, Vaasa, Finland. University of Vaasa.

Chankong, V. and Haimes, Y. Y. (1983). *Multiobjective Decision Making: Theory and Methodology.* Systems Science and Engineering. North-Holland.

Chipperfield, A. J. and Fleming, P. J. (1995). Gas Turbine Engine Controller Design using Multiobjective Genetic Algorithms. In Zalzala, A. M. S., editor, *Proceedings of the First IEE/IEEE International Conference on Genetic Algorithms in Engineering Systems : Innovations and Applications, GALESIA '95,* pages 214–219, Halifax Hall, University of Sheffield, UK. IEEE.

Coello, C. A., Christiansen, A. D., and Aguirre, A. H. (2000a). Use of Evolutionary Techniques to Automate the Design of Combinational Circuits. *International Journal of Smart Engineering System Design,* 2(4):299–314.

Coello, C. A. C. (1996). *An Empirical Study of Evolutionary Techniques for Multiobjective Optimization in Engineering Design.* PhD thesis, Department of Computer Science, Tulane University, New Orleans, LA.

Coello, C. A. C. (1999). A Comprehensive Survey of Evolutionary-Based Multiobjective Optimization Techniques. *Knowledge and Information Systems. An International Journal,* 1(3):269–308.

Coello, C. A. C. (2000a). Constraint-handling using an evolutionary multiobjective optimization technique. *Civil Engineering Systems,* 17:319–346.

Coello, C. A. C. (2000b). Handling Preferences in Evolutionary Multiobjective Optimization: A Survey. In *2000 Congress on Evolutionary Computation,* volume 1, pages 30–37, Piscataway, New Jersey. IEEE Service Center.

Coello, C. A. C. (2000c). Treating Constraints as Objectives for Single-Objective Evolutionary Optimization. *Engineering Optimization,* 32(3) :275–308.

Coello, C. A. C., Aguirre, A. H., and Buckles, B. P. (2000b). Evolutionary Multiobjective Design of Combinational Logic Circuits. In Lohn, J., Stoica, A., Keymeulen, D., and Colombano, S., editors, *Proceedings of the Second NASA/DoD Workshop on Evolvable Hardware,* pages 161–170, Los Alamitos, California. IEEE Computer Society.

Coello, C. A. C. and Christiansen, A. D. (1998). Two New GA-based methods for multiobjective optimization. *Civil Engineering Systems,* 15(3):207–243.

Coello, C. A. C., Christiansen, A. D., and Aguirre, A. H. (1998). Using a New GA-Based Multiobjective Optimization Technique for the Design of Robot Arms. *Robotica,* 16(4):401–414.

Coello, C. A. C. and Toscano, G. (2001). A Micro-Genetic Algorithm for Multiobjective Optimization. In Zitzler, E., Deb, K., Thiele, L., Coello, C. A. C., and Corne, D., editors, *First International Conference on Evolutionary Multi-Criterion Optimization,* pages 127–141. Springer-Verlag. Lecture Notes in Computer Science No. 1993.

Cohon, J. L. (1978). *Multiobjective Programming and Planning.* Academic Press.

Cohon, J. L. and Marks, D. H. (1975). A Review and Evaluation of Multiobjective Programming Techniques. *Water Resources Research,* 11(2):208–220.

Cvetković, D. and Parmee, I. C. (2000). Designer's preferences and multi-objective preliminary design processes. In Parmee, I. C., editor, *Proceedings of the Fourth International Conference on Adaptive Computing in Design and Manufacture (ACDM'2000),* pages 249–260. PEDC, University of Plymouth, UK, Springer London.

Czyzak, P. and Jaszkiewicz, A. (1997). Pareto Simulated Annealing. In Fandel, G. and Gal, T., editors, *Multiple Criteria Decision Making. Proceedings of the Xllth International Conference,* pages 297–307, Hagen, Germany. Springer-Verlag.

Das, I. and Dennis, J. (1997). A Closer Look at Drawbacks of Minimizing Weighted Sums of Objectives for Pareto Set Generation in Multicriteria Optimization Problems. *Structural Optimization,* 14(l):63–69.

Deb, K. (1999a). Multi–Objective Evolutionary Algorithms: Introducing Bias Among Pareto–Optimal Solutions. KanGAL report 99002, Indian Institute of Technology, Kanpur, India.

Deb, K. (1999b). Multi-Objective Genetic Algorithms: Problem Difficulties and Construction of Test Problems. *Evolutionary Computation,* 7(3):205–230.

Deb, K. (1999c). Solving Goal Programming Problems Using Multi-Objective Genetic Algorithms. In *1999 Congress on Evolutionary Computation,* pages 77–84, Washington, D.C. IEEE Service Center.

Deb, K. (2001). An Efficient Constraint Handling Method for Genetic Algorithms. *Computer Methods in Applied Mechanics and Engineering*. (in Press).

Deb, K., Agrawal, S., Pratab, A., and Meyarivan, T. (2000a). A Fast Elitist Non-Dominated Sorting Genetic Algorithm for Multi-Objective Optimization: NSGA-II. KanGAL report 200001, Indian Institute of Technology, Kanpur, India.

Deb, K., Agrawal, S., Pratab, A., and Meyarivan, T. (2000b). A Fast Elitist Non-Dominated Sorting Genetic Algorithm for Multi-Objective Optimization: NSGA-II. In *Proceedings of the Parallel Problem Solving from Nature VI Conference,* pages 849–858. Springer.

Deb, K. and Goldberg, D. E. (1989). An Investigation of Niche and Species Formation in Genetic Function Optimization. In Schaffer, J. D., editor, *Proceedings of the Third International Conference on Genetic Algorithms,* pages 42–50, San Mateo, California. George Mason University, Morgan Kaufmann Publishers.

Deb, K. and Meyarivan, T. (2000). Constrained Test Problems for Multi-Objective Evolutionary Optimization. KanGAL report 200005, Indian Institute of Technology, Kanpur, India.

Dick, R. P. and Jha, N. K. (1998). MOGAC: A Multiobjective Genetic Algorithm for Hardware-Software Co-synthesis of Hierarchical Heterogeneous Distributed Embedded Systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems,* 17(10) :920–935.

Duckstein, L. (1984). Multiobjective Optimization in Structural Design: The Model Choice Problem. In Atrek, E., Gallagher, R. H., Ragsdell, K. M., and Zienkiewicz, O. C., editors, *New Directions in Optimum Structural Design,* pages 459–481. John Wiley and Sons.

Edgeworth, F. Y. (1881). *Mathematical Physics.* P. Keagan, London, England.

Ehrgott, M. (2000). Approximation algorithms for combinatorial multicriteria optimization problems. *International Transactions in Operational Research,* 7:5–31.

Ehrgott, M. and Gandibleux, X. (2000). An Annotated Bibliography of Multi-objective Combinatorial Optimization. Technical Rep.- 62/2000, Fachbereich Mathematik, Universitat Kaiserslautern, Kaiserslautern, Germany.

Fonseca, C. M. and Fleming, P. J. (1993). Genetic Algorithms for Multiobjective Optimization: Formulation, Discussion and Generalization.

In Forrest, S., editor, *Proceedings of the Fifth International Conference on Genetic Algorithms,* pages 416–423, San Mateo, California. University of Illinois at Urbana-Champaign, Morgan Kauffman Publishers.

Fonseca, C. M. and Fleming, P. J. (1996). On the Performance Assessment and Comparison of Stochastic Multiobjective Optimizers. In Voigt, H.-M., Ebeling, W., Rechenberg, I., and Schwefel, H.-P., editors, *Parallel Problem Solving from Nature—PPSNIV,* Lecture Notes in Computer Science, pages 584–593, Berlin, Germany. Springer-Verlag.

Fonseca, C. M. and Fleming, P. J. (1998). Multiobjective Optimization and Multiple Constraint Handling with Evolutionary Algorithms—Part I: A Unified Formulation. *IEEE Transactions on Systems, Man, and Cybernetics, Part A: Systems and Humans,* 28(l):26–37.

Gandibleux, X., Mezdaoui, N., and Fréville, N. (1997). A tabu search procedure to solve combinatorial optimisation problems. In Caballero, R., Ruiz, F., and Steuer, R., editors, *Advances in Multiple Objective and Goal Programming,* volume 455 of *Lecture Notes in Economics and Mathematical Systems,* pages 291–300. Springer-Verlag.

Gen, M. and Cheng, R. (1997). *Genetic Algorithms and Engineering Design.* John Wiley and Sons, Inc., New York.

Goicoechea, A., Duckstein, L., and Fogel, M. (1979). Multiple objectives under uncertainty: An illustrative application of PROTRADE. *Water Resources Research,* 15(2):203–210.

Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison-Wesley Publishing Company, Reading, Massachusetts.

Greenwood, G. W., Hu, X. S., and D'Ambrosio, J. G. (1997). Fitness Functions for Multiple Objective Optimization Problems: Combining Preferences with Pareto Rankings. In Belew, R. K. and Vose, M. D., editors, *Foundations of Genetic Algorithms 4,* pages 437–455, San Mateo, California. Morgan Kaufmann.

Habenicht, W. (1982). Quad trees, A data structure for discrete vector optimization problems. In *Lecture notes in economics and mathematical systems,* volume 209, pages 136–145.

Hajela, P. and Lin, C. Y. (1992). Genetic search strategies in multicriterion optimal design. *Structural Optimization,* 4:99–107.

Hanne, T. (2000). On the convergence of multiobjective evolutionary algorithms. *European Journal of Operational Research,* 117(3):553–564.

Hansen, M. P. (1996). Tabu Search in Multiobjective Optimisation : MOTS. In *Proceedings of MCDM'97,* Cape Town, South Africa.

Hartmann, J. W., Coverstone-CarrolI, V., and Williams, S. N. (1998). Optimal interplanetary spacecraft trajectories via a pareto genetic algorithm. *Journal of the Astronautical Sciences,* 46(3):267–282.

Hilliard, M. R., Liepins, G. E., Palmer, M., and Rangarajen, G. (1989). The computer as a partner in algorithmic design: Automated discovery of parameters for a multiobjective scheduling heuristic. In Sharda, R., Golden, B. L., Wasil, E., Balci, O., and Stewart, W., editors, *Impacts of Recent Computer Advances on Operations Research.* North-Holland Publishing Company, New York.

Hinterding, R. and Michalewicz, Z. (1998). Your Brains and My Beauty: Parent Matching for Constrained Optimisation. In *Proceedings of the 5th International Conference on Evolutionary Computation,* pages 810–815, Anchorage, Alaska.

Horn, J. (1997). Multicriterion Decision Making. In Bäck, T., Fogel, D., and Michalewicz, Z., editors, *Handbook of Evolutionary Computation,* volume 1, pages F1.9:l – F1.9:15. IOP Publishing Ltd. and Oxford University Press.

Horn, J., Nafpliotis, N., and Goldberg, D. E. (1994). A Niched Pareto Genetic Algorithm for Multiobjective Optimization. In *Proceedings of the First IEEE Conference on Evolutionary Computation, IEEE World Congress on Computational Intelligence,* volume 1, pages 82–87, Piscataway, New Jersey. IEEE Service Center.

Hwang, C. L. and Masud, A. S. M. (1979). Multiple Objective Decision-Making Methods and Applications. In *Lecture Notes in Economics and Mathematical Systems,* volume 164. Springer-Verlag, New York.

Hwang, C. L., Paidy, S. R., and Yoon, K. (1980). Mathematical Programming with Multiple Objectives: A Tutorial. *Computing and Operational Research,* 7:5–31.

Ishibuchi, H. and Murata, T. (1996). Multi-Objective Genetic Local Search Algorithm. In Fukuda, T. and Furuhashi, T., editors, *Proceedings of the 1996 International Conference on Evolutionary Computation,* pages 119–124, Nagoya, Japan. IEEE.

Jakob, W., Gorges-Schleuter, M., and Blume, C. (1992). Application of Genetic Algorithms to task planning and learning. In Männer, R. and Manderick, B., editors, *Parallel Problem Solving from Nature, 2nd Workshop,* Lecture Notes in Computer Science, pages 291–300, Amsterdam. North-Holland Publishing Company.

Jaszkiewicz, A. (2000). On the performance of multiple objective genetic local search on the 0/1 knapsack problem, a comparative experiment. Technical Report RA-002/2000, Institute of Computing Science, Poznan University of Technology, Poznań, Poland.

Jiménez, F., Verdegay, J. L., and Gómez-Skarmeta, A. F. (1999). Evolutionary Techniques for Constrained Multiobjective Optimization Problems. In Wu, A. S., editor, *Proceedings of the 1999 Genetic and Evolutionary Computation Conference. Workshop Program,* pages 115–116, Orlando, Florida.

Karlin, S. (1959). Mathematical Methods and Theory in Games. In *Programming and Economics,* volume 1, pages 216–217. Addison-Wesley, Reading, Massachusetts.

Knowles, J. D. and Corne, D. W. (1999). The Pareto Archived Evolution Strategy: A New Baseline Algorithm for Multiobjective Optimisation. In *1999 Congress on Evolutionary Computation,* pages 98–105, Washington, D.C. IEEE Service Center.

Knowles, J. D. and Corne, D. W. (2000). Approximating the Nondominated Front Using the Pareto Archived Evolution Strategy. *Evolutionary Computation,* 8(2):149–172.

Koopman, B. O. (1953). The Optimum Distribution of Effort. *Operations Research,* l(2):52–63.

Koopmans, T. C. (1951). Analysis of Production as an efficient combination of activities. In Koopmans, T. C., editor, *Activity Analysis of Production and Allocation, Cowles Commision Monograph No. 13,* pages 33–97. John Wiley and Sons, New York, New York.

Krishnakumar, K. (1989). Micro-genetic algorithms for stationary and non-stationary function optimization. In *SPIE Proceedings: Intelligent Control and Adaptive Systems,* pages 289–296.

Kuhn, H. W. and Tucker, A. W. (1951). Nonlinear programming. In Neyman, J., editor, *Proceedings of the Second Berkeley Symposium on Mathematical Statistics and Probability,* pages 481–492, Berkeley, California. University of California Press.

Kurahashi, S. and Terano, T. (2000). A Genetic Algorithm with Tabu Search for Multimodal and Multiobjective Function Optimization. In Whitley, D., Goldberg, D., Cantú-Paz, E., Spector, L., Parmee, I, and Beyer, H.-G., editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'2000),* pages 291–298, San Francisco, California. Morgan Kaufmann.

Liu, X., Begg, D. W., and Fishwick, R. J. (1998). Genetic approach to optimal topology/controller design of adaptive structures. *International Journal for Numerical Methods in Engineering,* 41:815–830.

Loucks, D. P. (1975). Conflict and choice: Planning for multiple objectives. In Blitzer, C., Clark, P., and Taylor, L., editors, *Economy wide Models and Development Planning,* New York, New York. Oxford University Press.

Marglin, S. (1967). *Public Investment Criteria.* MIT Press, Cambridge, Massachusetts.

Massebeuf, S., Fonteix, C., Kiss, L. N., Marc, I., Pla, F., and Zaras, K. (1999). Multicriteria Optimization and Decision Engineering of an Extrusion Process Aided by a Diploid Genetic Algorithm. In *1999 Congress on Evolutionary Computation,* pages 14–21, Washington, D.C. IEEE Service Center.

Michalewicz, Z. (1996). *Genetic Algorithms + Data Structures = Evolution Programs.* Springer-Verlag, third edition.

Miettinen, K. M. (1998). *Nonlinear Multiobjective Optimization.* Kluwer Academic Publishers, Boston, Massachusetts.

Mitchell, M. (1996). *An Introduction to Genetic Algorithms.* MIT Press, Cambridge, Massachusetts.

Mitra, K., Deb, K., and Gupta, S. K. (1998). Multiobjective Dynamic Optimization of an Industrial Nylon 6 Semibatch Reactor Using Genetic Algorithm. *Journal of Applied Polymer Science,* 69(l):69–87.

Monarchi, D. E., Kisiel, C. C., and Duckstein, L. (1973). Interactive Multiobjective Programming in Water Resources: A Case Study. *Water Resources Research,* 9(4):837–850.

Narayanan, S. and Azarm, S. (1999). On Improving Multiobjective Genetic Algorithms for Design Optimization. *Structural Optimization,* 18:146–155.

Pareto, V. (1896). *Cours D'Economie Politique,* volume I and II. F. Rouge, Lausanne.

Parks, G. T. and Miller, I. (1998). Selective Breeding in a Multiobjective Genetic Algorithm. In Eiben, A. E., Schoenauer, M., and Schwefel, H.-P., editors, *Parallel Problem Solving From Nature — PPSN V,* pages 250–259, Amsterdam, Holland. Springer-Verlag.

Parmee, I. C. and Purchase, G. (1994). The development of a directed genetic search technique for heavily constrained design spaces. In Parmee, I. C., editor, *Adaptive Computing in Engineering Design and Control-'94* pages 97–102, Plymouth, UK. University of Plymouth, University of Plymouth.

Pierreval, H. and Plaquin, M.-F. (1998). An Evolutionary Approach of Multicriteria Manufacturing Cell Formation. *International Transactions in Operational Research,* 5(l):13–25.

Quagliarella, D. and Vicini, A. (1997). Coupling Genetic Algorithms and Gradient Based Optimization Techniques. In Quagliarella, D., Périaux, J., Poloni, C., and Winter, G., editors, *Genetic Algorithms and Evolution Strategies in Engineering and Computer Science. Recent Advances and Industrial Applications,* chapter 14, pages 289–309. John Wiley and Sons, West Sussex, England.

Ray, T., Kang, T., and Chye, S. K. (2000). An Evolutionary Algorithm for Constrained Optimization. In Whitley, D., Goldberg, D., Cantú-Paz, E., Spector, L., Parmee, I., and Beyer, H.-G., editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'2000),* pages 771–777, San Francisco, California. Morgan Kaufmann.

Rekiek, B., Lit, P. D., Pellichero, F., L'Eglise, T., Falkenauer, E., and Delchambre, A. (2000). Dealing With User's Preferences in Hybrid Assembly Lines Design. In *Proceedings of the MCPL'2000 Conference.*

Richardson, J. T., Palmer, M. R., Liepins, G., and Hilliard, M. (1989). Some Guidelines for Genetic Algorithms with Penalty Functions. In Schaffer, J. D., editor, *Proceedings of the Third International Conference on Genetic Algorithms,* pages 191–197, George Mason University. Morgan Kaufmann Publishers.

Ritzel, B. J., Eheart, J. W., and Ranjithan, S. (1994). Using genetic algorithms to solve a multiple objective groundwater pollution containment problem. *Water Resources Research,* 30(5): 1589–1603.

Romero, C. E. M. and Manzanares, E. M. (1999). MOAQ an Ant-Q Algorithm for Multiple Objective Optimization Problems. In Banzhaf, W., Daida, J., Eiben, A. E., Garzon, M. H., Honavar, V., Jakiela, M., and Smith, R. E., editors, *Genetic and Evolutionary Computing Conference (GECCO 99),* volume 1, pages 894–901, San Francisco, California. Morgan Kaufmann.

Rosenberg, R. S. (1967). *Simulation of genetic populations with biochemical properties.* PhD thesis, University of Michigan, Ann Harbor, Michigan.

Rudolph, G. (1998). On a Multi-Objective Evolutionary Algorithm and Its Convergence to the Pareto Set. In *Proceedings of the 5th IEEE Conference on Evolutionary Computation,* pages 511–516, Piscataway, New Jersey. IEEE Press.

Rudolph, G. and Agapie, A. (2000). Convergence Properties of Some Multi-Objective Evolutionary Algorithms. In *Proceedings of the 2000 Conference on Evolutionary Computation,* volume 2, pages 1010–1016, Piscataway, New Jersey. IEEE Press.

Sandgren, E. (1994). Multicriteria design optimization by goal programming. In Adeli, H., editor, *Advances in Design Optimization,* chapter 23, pages 225–265. Chapman & Hall, London.

Schaffer, J. D. (1985). Multiple Objective Optimization with Vector Evaluated Genetic Algorithms. In *Genetic Algorithms and their Applications: Proceedings of the First International Conference on Genetic Algorithms,* pages 93–100. Lawrence Erlbaum.

Schroder, P., Chipperfield, A. J., Fleming, P. J., and Grum, N. (1997). Multi-Objective Optimization of Distributed Active Magnetic Bearing Controllers. In *Genetic Algorithms in Engineering Systems: Innovations and Applications,* pages 13–18. IEE.

Shaw, K. J. and Fleming, P. J. (1997). Including Real-Life Preferences in Genetic Algorithms to Improve Optimisation of Production Schedules. In *Proceedings of the GALESIA '97,* Glasgow, Scotland. IEE.

Srinivas, N. and Deb, K. (1994). Multiobjective Optimization Using Nondominated Sorting in Genetic Algorithms. *Evolutionary Computation,* 2(3):221–248.

Stadler, W. (1975). Preference optimality and applications to Pareto optimality. In Leitmann, G. and Marzollo, A., editors, *Multi-Criteria Decision Making,* volume 211. Springer-Verlag, New York.

Stadler, W. (1988). Fundamentals of multicriteria optimization. In Stadler, W., editor, *Multicriteria Optimization in Engineering and the Sciences,* pages 1–25. Plenum Press, New York.

Surry, P. D. and Radcliffe, N. J. (1997). The COMOGA Method: Constrained Optimisation by Multiobjective Genetic Algorithms. *Control and Cybernetics,* 26(3).

Surry, P. D., Radcliffe, N. J., and Boyd, I. D. (1995). A Multi-Objective Approach to Constrained Optimisation of Gas Supply Networks : The COMOGA Method. In Fogarty, T. C., editor, *Evolutionary Computing. AISB Workshop. Selected Papers,* Lecture Notes in Computer Science, pages 166–180, Sheffield, U.K. Springer-Verlag.

Tan, K. C. and Li, Y. (1997). Multi-Objective Genetic Algorithm Based Time and Frequency Domain Design Unification of Linear Control Systems. Technical Report CSC-97007, Department of Electronics and Electrical Engineering, University of Glasglow, Glasglow, Scotland.

Tanino, T., Tanaka, M., and Hojo, C. (1993). An interactive multicriteria decision making method by using a genetic algorithm. In *Proceedings of 2nd International Conference on Systems Science and Systems Engineering,* pages 381–386.

van Huylenbroeck, G. (1995). The Conflict Analysis Method: bridging the gap between ELECTRE, PROMETHEE and ORESTE. *European Journal of Operational Research,* 82(3):490–502.

Vedarajan, G., Chan, L. C., and Goldberg, D. E. (1997). Investment Portfolio Optimization using Genetic Algorithms. In Koza, J. R., editor, *Late Breaking Papers at the Genetic Programming 1997 Conference,* pages 255–263, Stanford University, California. Stanford Bookstore.

Veldhuizen, D. A. V. (1999). *Multiobjective Evolutionary Algorithms: Classifications, Analyses, and New Innovations.* PhD thesis, Depart-

ment of Electrical and Computer Engineering. Graduate School of Engineering. Air Force Institute of Technology, Wright-Patterson AFB, Ohio.

Veldhuizen, D. A. V. and Lamont, G. B. (1998). Evolutionary Computation and Convergence to a Pareto Front. In Koza, J. R., editor, *Late Breaking Papers at the Genetic Programming 1998 Conference,* pages 221–228, Stanford University, California. Stanford University Bookstore.

Veldhuizen, D. A. V. and Lamont, G. B. (1999). Multiobjective Evolutionary Algorithm Test Suites. In Carroll, J., Haddad, H., Oppenheim, D., Bryant, B., and Lamont, G. B., editors, *Proceedings of the 1999 ACM Symposium on Applied Computing,* pages 351–357, San Antonio, Texas. ACM.

Vincke, P. (1995). Analysis of MCDA in Europe. *European Journal of Operational Research,* 25:160–168.

von Neumann, J. and Morgenstern, O. (1944). *Theory of Games and Economic Behavior.* Princeton University Press, Princeton, New Jersey.

Wienke, P. B., Lucasius, C., and Kateman, G. (1992). Multicriteria target optimization of analytical procedures using a genetic algorithm. *Analytical Chimica Acta,* 265(2) :211–225.

Wilson, P. B. and Macleod, M. D. (1993). Low implementation cost IIR digital filter design using genetic algorithms. In *IEE/IEEE Workshop on Natural Algorithms in Signal Processing,* pages 4/1–4/8, Chelmsford, U.K.

Zadeh, L. A. (1963). Optimality and Nonscalar-Valued Performance Criteria. *IEEE Transactions on Automatic Control,* AC-8(1):59–60.

Zebulum, R. S., Pacheco, M. A., and Vellasco, M. (1998). A multiobjective optimisation methodology applied to the synthesis of low-power operational amplifiers. In Cheuri, I. J. and dos Reis Filho, C. A., editors, *Proceedings of the XIII International Conference in Microelectronics and Packaging,* volume 1, pages 264–271, Curitiba, Brazil.

Zitzler, E., Deb, K., and Thiele, L. (2000). Comparison of Multiobjective Evolutionary Algorithms: Empirical Results. *Evolutionary Computation,* 8(2):173–195.

Zitzler, E., Teich, J., and Bhattacharyya, S. S. (1999). Multidimensional Exploration of Software Implementations for DSP Algorithms. *VLSI Signal Processing Systems.* (To appear).

Zitzler, E. and Thiele, L. (1999). Multiobjective Evolutionary Algorithms: A Comparative Case Study and the Strength Pareto Approach. *IEEE Transactions on Evolutionary Computation,* 3(4):257–271.

Chapter 6

MULTI-OBJECTIVE EVOLUTIONARY ALGORITHMS FOR ENGINEERING SHAPE DESIGN

Kalyanmoy Deb and
Tushar Goel

Abstract Evolutionary optimization algorithms work with a population of solutions, instead of a single solution. Since multi-objective optimization problems give rise to a set of Pareto-optimal solutions, evolutionary optimization algorithms are ideal for handling multi-objective optimization problems. Many years of research and application studies have produced a number of efficient multi-objective evolutionary algorithms (MOEAs), which are ready to be applied to real-world problems. In this paper, we propose a practical approach, which will enable an user to find a set of non-dominated solutions closer to the true Pareto-optimal front and simultaneously reduce the size of the obtained non-dominated solution set. The efficacy of the proposed approach is demonstrated in solving a number of mechanical shape optimization problems, including a simply-supported plate design, a cantilever plate design, a hoister design, and a bicycle frame design. The results are interesting and suggest immediate application of the proposed technique to more complex engineering design problems.

Keywords: Genetic algorithms, multi-objective optimization, shape optimization, Pareto-optimum, mechanical component design.

1. Introduction

During the last decade, a number of multi-objective optimization techniques using evolutionary algorithms are suggested (Deb et al., 2000b; Horn et al., 1994; Fonseca and Fleming, 1993; Knowles and Corne, 1999; Srinivas and Deb, 1995; Zitzler and Thiele, 1998). The outcome of these studies is that different multi-objective optimization problems are possible to solve for the purpose of finding multiple Pareto-optimal

solutions in one *single* simulation run. Classical means of finding one solution at a time with a weight vector or with a similar approach requires apriori knowledge of weight vector and need to be run many times, hopefully finding a different Pareto-optimal solution each time. In addition to converging close or on the true Pareto-optimal set, multi-objective evolutionary algorithms (MOEAs) are capable to finding a widely distributed set of solutions.

In this paper, we suggest a hybrid technique to take evolutionary multi-objective optimization procedures one step closer to practice. Specifically, in a real-world problem, we would like to ensure a better convergence to the true Pareto-optimal front and would also like to reduce the size of obtained non-dominated solutions to a reasonable number. The solutions obtained by an MOEA are modified using a local search method, in which a weighted objective function is minimized. The use of a local search method from the MOEA solutions will allow a better convergence to the true Pareto-optimal front. A clustering method is suggested to reduce the size of the obtained set of solutions. For finite search space problems, the local search approach may itself reduce the size of the obtained set.

A specific MOEA—elitist non-dominated sorting GA or NSGA-II—and a hill-climbing local search method are used together to solve a number of engineering shape optimization problems for two objectives. Minimizing the weight of a structure and minimizing the maximum deflection of the structure have conflicting solutions. When these two objectives are considered together in a design, a number of Pareto-optimal solutions result. By representing presence and absence of small constituting elements in a binary string (Chapman and Jakiela, 1996; Chapman et al., 1994; Duda and Jakiela, 1997), NSGA-II uses an innovative crossover operator which seems to help in combining good partial solutions together to form bigger partial solutions. The finite element method is used to evaluate a string representing a shape. The paper shows how the proposed hybrid technique can find a number of solutions with different trade-offs between weight and deflection. On a cantilever plate design, a simply-supported plate design, a hoister plate design, and a bicycle frame design problem, the proposed technique finds interesting and well-engineered solutions. These results indicate that the proposed hybrid technique is ready to be applied to more complex engineering shape design problems.

2. Multi-Objective Optimization and Pareto-Optimality

The principles of multi-objective optimization are different from that of a single-objective optimization. The main goal in a single-objective optimization is to find the global optimal solution. However, in a multi-objective optimization problem, there are more than one objective function, each of which may have a *different* individual optimal solution. If there is sufficient difference in the optimal solutions corresponding to different objectives, the objective functions are often known as *conflicting* to each other. Multi-objective optimization with such conflicting objective functions gives rise to a set of optimal solutions, instead of one optimal solution. The reason for the optimality of many solutions is that no one solution can be considered to be better than any other with respect to all objective functions. These optimal solutions have a special name—Pareto-optimal solutions.

It is clear that the concept of optimality in multi-objective optimization deals with a number (or a set) of solutions, instead of one solution. Based on the above discussions, we first define conditions for a solution to become dominated with respect to another solution and then present conditions for a set of solutions to become a Pareto-optimal set.

For a problem having more than one objective function (say, f_j, $j = 1, \ldots, M$ and $M > 1$), any two solutions $x^{(1)}$ and $x^{(2)}$ (having P decision variables each) can have one of two possibilities—one dominates the other or none dominates the other. A solution $x^{(1)}$ is said to dominate the other solution $x^{(2)}$, if both the following conditions are true (Steuer, 1986):

1 The solution $x^{(1)}$ is no worse (say the operator $\prec$ denotes worse and $\succ$ denotes better) than $x^{(2)}$ in all objectives, or $f_j(x^{(1)}) \not\prec f_j(x^{(2)})$ for all $j = 1, 2, \ldots, M$ objectives.

2 The solution $x^{(1)}$ is strictly better than $x^{(2)}$ in at least one objective, or $f_{\bar{j}}(x^{(1)}) \succ f_{\bar{j}}(x^{(2)})$ for at least one $\bar{j} \in \{1, 2, \ldots, M\}$.

If any of the above conditions is violated, the solution $x^{(1)}$ does not dominate the solution $x^{(2)}$. If $x^{(1)}$ dominates the solution $x^{(2)}$, it is also customary to write $x^{(2)}$ is dominated by $x^{(1)}$, or $x^{(1)}$ is non-dominated by $x^{(2)}$, or, simply, among the two solutions, $x^{(1)}$ is the non-dominated solution. Although the above is a standard definition of domination among two solutions, Parmee et al. (Parmee et al., 2000) suggested a weighted dominance relation based on relative importance of objectives. In problems where a weight information is available, such modified definitions may be useful.

The following definitions ensure whether a set of solutions belong to a local or global Pareto-optimal set, similar to the definitions of local and global optimal solutions in single-objective optimization problems:

Local Pareto-optimal Set: If for every member x in a set $\underline{P}$, there exist no solution y satisfying $\|y - x\|_\infty \leq \epsilon$, where ϵ is a small positive number (in principle, y is obtained by perturbing x in a small neighborhood), which dominates any member in the set $\underline{P}$, then the solutions belonging to the set $\underline{P}$ constitute a local Pareto-optimal set.

Global Pareto-optimal Set: If there exists no solution in the search space which dominates any member in the set $\bar{P}$, then the solutions belonging to the set $\bar{P}$ constitute a global Pareto-optimal set.

We would like to highlight here that there exists a difference between a non-dominated set and a Pareto-optimal set. A non-dominated set is defined in the context of a sample of the search space. In a sample of search points, solutions that are not dominated (according to the above definition) by any other solution in the sample space are non-dominated solutions. A Pareto-optimal set is a non-dominated set, when the sample is the entire search space.

From the above discussions, we observe that there are primarily two goals that a multi-objective optimization algorithm must try to achieve:

1 Guide the search towards the global Pareto-optimal region, and

2 Maintain population diversity in the Pareto-optimal front.

The first task is a natural goal of any optimization algorithm. The second task is unique to multi-objective optimization only. Since no one solution in the Pareto-optimal set can be said to be better than the other, what an algorithm can do best to find as many different Pareto-optimal solutions as possible.

The classical ways of tackling multi-objective optimization problems is straightforward: convert multiple objectives into one objective. There exists a number of conversion methods (Chankong and Haimes, 1983; Miettinen, 1999; Sen and Yang, 1998)—weighted sum approach, ϵ- perturbation method, Tchebycheff method, min-max method, goal programming method and others. Since multiple objectives are converted into one objective, the resulting solution to the single-objective optimization problem is usually subjective to the parameter settings chosen by the user. Since a classical optimization method is used to solve the resulting problem, only one solution (hopefully a Pareto-optimal solution) can be

found in one simulation run. Thus, in order to find multiple Pareto-optimal solutions, the chosen optimization algorithm must have to be used a number of times. Furthermore, the classical methods have been found to be sensitive to the convexity and continuity of the Pareto-optimal region (Deb, 2001).

3. Elitist Non-dominated Sorting GA (NSGA-II)

The details of NSGA-II algorithm appears elsewhere (Deb et al., 2000a). Initially, a random parent population P_0 is created. The population is sorted based on the non-domination. A special book keeping procedure is used in order to reduce the computational complexity down to $O(MN^2)$. Each solution is assigned a fitness equal to its non-domination level. Binary tournament selection, recombination, and mutation operators are used to create a child population Q_0 of size *N*. Thereafter, we use the following algorithm in every generation.

$R_t = P_t \cup Q_t$
$\mathcal{F} =$ `fast-nondominated-sort(`R_t`)`
$P_{t+1} = \emptyset$ and $i = 1$
until $|P_{t+1}| + |\mathcal{F}_i| \leq N$
    `crowding-distance-assignment(`$\mathcal{F}_i$`)`
    $P_{t+1} = P_{t+1} \cup \mathcal{F}_i$
    $i = i + 1$
Sort$(\mathcal{F}_i, \prec_n)$
$P_{t+1} = P_{t+1} \cup \mathcal{F}_i[1 : (N - |P_{t+1}|)]$
$Q_{t+1} =$ `make-new-pop(`P_{t+1}`)`
$t = t + 1$

First, a combined population $R_t = P_t \cup Q_t$ is formed. This allows parent solutions to be compared with the child population, thereby ensuring elitism. The population R_t is of size 2*N*. Then, the population R_t is sorted according to non-domination (Steuer, 1986). The sorting procedure classifies the population into several non-dominated fronts, $\mathcal{F}_\infty$, $\mathcal{F}_\in$, and so on. The new parent population P_{t+1} is formed by adding solutions from the first front $\mathcal{F}_\infty$ and continuing to other fronts successively till the size exceeds *N*. Thereafter, the solutions of the last accepted front are sorted according to a crowded distance metric and a total of *N* points are picked. The crowded distance is measured as the perimeter of the maximum box containing the two neighboring solutions in the objective space, as depicted in Figure 6.1. For details, refer to the original study (Deb et al., 2000a). Since the diversity among the

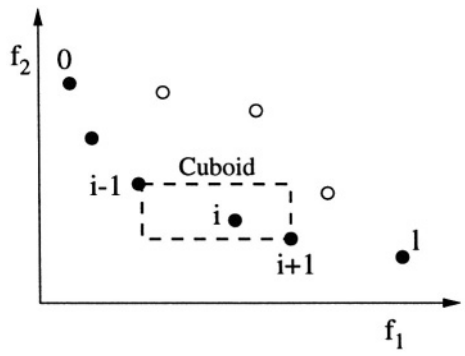


Figure 6.1. The crowding distance calculation is shown.

solutions is important, we define a *crowded comparison operator* using a relation $\prec_n$ as follows:

Definition 1 *Solution i is better than solution j in relation $\prec_n$ if ($i_{rank} < j_{rank}$) or (($i_{rank} = j_{rank}$) and ($i_{distance} > j_{distance}$)).*

In the tournament selection, we use the above crowded comparison operator to choose one of the two solutions. Between two solutions with differing non-domination ranks in a tournament, we prefer the solutions with the lower rank. Otherwise, if both the points belong to the same front then we prefer the point which is located in a region with lesser number of points (or with larger crowded distance). This way solutions from less dense regions in the search space are given importance in deciding which solutions to choose from R_t. This constructs the population P_{t+1}. This population of size N is now used for selection, crossover and mutation to create a new population Q_{t+1} of size N. We use a binary tournament selection operator but the selection criterion is now based on the crowded comparison operator $\prec_n$. The above procedure is continued for a specified number of generations.

It is clear from the above description that NSGA-II uses (i) a faster non-dominated sorting approach, (ii) an elitist strategy, and (ii) no niching parameter. It has been shown elsewhere (Deb, 2001) that the above procedure has $O(MN^2)$ computational complexity.

3.1 Two Test Problems

We illustrate the working of NSGA-II on two difficult test problems. The function ZDT2 has a non-convex Pareto-optimal region, whereas the second function KUR has a discontinuous Pareto-optimal region (Deb,

2001):

$$\text{ZDT2:} \quad \begin{cases} f_1(\mathbf{x}) = x_1, \\ f_2(\mathbf{x}) = g(\mathbf{x}) \left[1 - (x_1/g(\mathbf{x}))^2\right], \\ g(\mathbf{x}) = 1 + 9 \left(\sum_{i=2}^{30} x_i\right)/29, \\ 0 \le x_i \le 1, \quad i = 1, 2, \ldots, 30. \end{cases} \tag{6.1}$$

$$\text{KUR:} \quad \begin{cases} f_1(\mathbf{x}) = \sum_{i=1}^{3} \left(-10 \exp\left(-0.2\sqrt{x_i^2 + x_{i+1}^2}\right)\right), \\ f_2(\mathbf{x}) = \sum_{i=1}^{3} \left(|x_i|^{0.8} + 5 \sin x_i^3\right) \\ -5 \le x_i \le 5, \quad i = 1, 2, 3. \end{cases} \tag{6.2}$$

Figure 6.2 shows the obtained non-dominated set of solutions for ZDT2. All solutions lie on the true Pareto-optimal front. Binary-coded GAs with a single-point crossover operator ($p_c = 0.9$) and a bit-wise mutation operator ($p_m = 1/600$) are used. For each variable, we use 20 bits. A population of size 100 is chosen. NSGA-II is run for 250 generations. It is clear that NSGA-II is able to maintain a wide spread of solutions on the Pareto-optimal front, which is non-convex. The search space lies above the Pareto-optimal front shown in the figure.

Figure 6.3 shows that NSGA-II is also able to find a well-distributed set of solutions in all three discontinuous regions of Pareto-optimal front for KUR. Here, a real-coded GA is used. Variables are coded directly

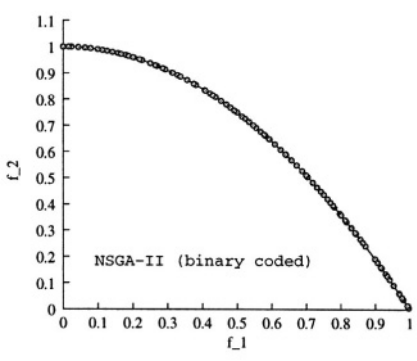


Figure 6.2. Non-dominated solutions with NSGA-II (binary-coded) on ZDT2 are shown.

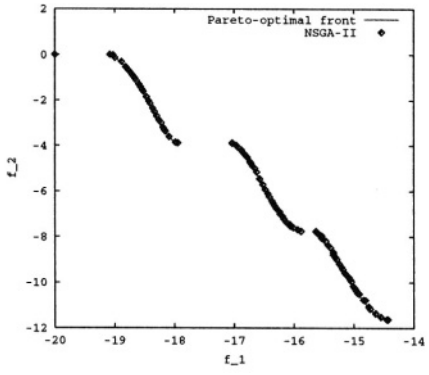


Figure 6.3. Non-dominated solutions with NSGA-II (real-coded) on KUR are shown.

and the simulated binary crossover operator with $p_c = 0.9$ (Deb and Agrawal, 1995) and a polynomial mutation operator with $p_m = 1/3$ are used. NSGA-II with a population size of 100 is run for a maximum of 250

generations. Despite disconnected regions, the NSGA-II can find well-distributed solutions in all regions. The above two results demonstrate the NSGA-II procedure can be used with binary-coded as well as real-coded implementations.

3.2 Constraint handling

The usual way of handling constraints using penalty functions is criticized for their sensitivity on the so-called penalty parameters. Here, we suggest a more elegant approach for constraint handling as follows. We simply change the definition of domination between two solutions as follows:

Definition 2 *A solution i is said to constrained-dominate a solution j, if any of the following conditions is true:*

1 *Solution i is feasible and solution j is not.*

2 *Solutions i and j are both infeasible, but solution i has a smaller constraint violation.*

3 *Solutions i and j are feasible and solution i dominates solution j.*

This way, feasible solutions *constrained-dominate* any infeasible solution and two infeasible solutions are compared based on their constraint violations only. However, when two feasible solutions are compared, they are checked based on their usual domination level.

3.2.1 Simulation Results. We illustrate the working of the constraint handling strategy on the following problem:

$$\text{TNK:} \begin{cases} f_1(\mathbf{x}) = x_1, \\ f_2(\mathbf{x}) = x_2, \\ g_1(\mathbf{x}) = -x_1^2 - x_2^2 + 1 + 0.1\cos(16\arctan(x_1/x_2)) \leq 0, \\ g_2(\mathbf{x}) = (x_1 - 0.5)^2 + (x_2 - 0.5)^2 \leq 0.5, \\ 0 \leq x_i \leq \pi, \quad i = 1, 2. \end{cases} \tag{6.3}$$

Figure 6.4 shows the NSGA-II population after 100 generations. Real-coded NSGA-II with identical parameter setting as in KUR is used here. It is clear that NSGA-II is able to distribute population members on the critical constraint boundary, portions of which form the constrained Pareto-optimal set. Regions which are not Pareto-optimal are not found by the NSGA-II. This shows the ability of NSGA-II along with the constraint-domination principle in converging and in distributing solutions on the true Pareto-optimal regions.

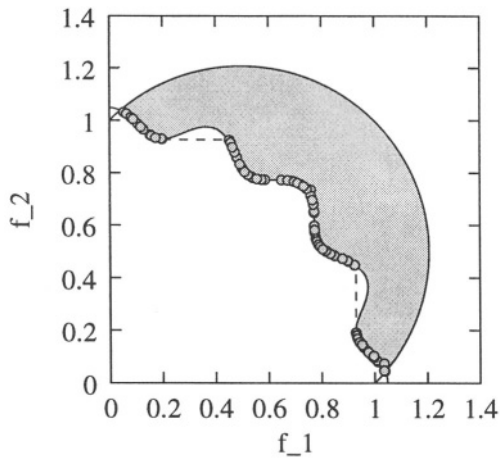


Figure 6.4. Obtained non-dominated solutions with NSGA-II on the constrained problem TNK are shown.

4. Hybrid Approach

It is clear from the previous section that NSGA-II is an efficient procedure of finding a wide-spread as well as well-converged set of solutions in a multi-objective optimization problem. All problems considered in the previous section and in earlier studies are test problems where exact location of Pareto-optimal solutions is known. Such test problems are necessary to evaluate the working of any MOEA. However, in this section we would like to take MOEAs a step closer to practice by

1 ensuring convergence closer to the true Pareto-optimal front, and

2 reducing the size of the obtained non-dominated set.

We illustrate both the above issues in the following subsections.

4.1 Converging better

In a real-world problem, the knowledge of the Pareto-optimal front is usually not known. Although NSGA-II has demonstrated good convergence properties in test problems, we enhance the probability of its convergence by using a hybrid approach. A local search strategy is suggested from each obtained solution of NSGA-II to find a better solution. Since a local search strategy requires a single objective function, a weighted objective or a Tchebycheff metric or any other metric which

will convert multiple objectives into a single objective can be used. In this study, we use a weighted objective:

$$F(\mathbf{x}) = \sum_{j=1}^{M} \bar{w}_j^{\mathbf{x}} f_j(\mathbf{x}), \tag{6.4}$$

where weights are calculated from the obtained set of solutions in a special way. First, the minimum $f_j^{\min}$ and maximum $f_j^{\max}$ values of each objective function f_j are noted. Thereafter, for any solution $\mathbf{x}$ in the obtained set, the weight for each objective function is calculated as follows:

$$\bar{w}_j^{\mathbf{x}} = \frac{(f_j^{\max} - f_j(\mathbf{x}))/(f_j^{\max} - f_j^{\min})}{\sum_{k=1}^{M}(f_k^{\max} - f_k(\mathbf{x}))/(f_k^{\max} - f_k^{\min})}. \tag{6.5}$$

In the above calculation, minimization of objective functions is assumed. When a solution $\mathbf{x}$ is close to the individual minimum of the function f_j, the numerator becomes one, causing a large value of the weight for this function. For an objective which has to be maximized, the term $(f_j^{\max} - f_j(\mathbf{x}))$ needs to be replaced with $(f_j(\mathbf{x}) - f_j^{\min})$. The division of the numerator with the denominator ensures that the calculated weights are normalized or $\sum_{j=1}^{M} \bar{w}_j^{\mathbf{x}} = 1$.

In order to distinguish the above calculated weight from the usual user-specified weight needed in weighted multi-objective optimization algorithms, we call these calculated weights $\bar{\mathbf{w}}$ as pseudo-weights.

Once the pseudo-weights are calculated, the local search procedure is simple. Begin the search from each solution $\mathbf{x}$ independently with the purpose of optimizing $F(\mathbf{x})$. Figure 6.5 illustrates this procedure. Since, the pseudo-weight vector $\bar{\mathbf{w}}$ dictates roughly the priority of different objective functions at that solution, optimizing $F(\mathbf{x})$ will produce a Pareto-optimal or a near Pareto-optimal solution. This is true for convex Pareto-optimal regions. However, for non-convex Pareto-optimal regions, there exists no weight vector corresponding to Pareto-optimal solutions in certain regions. Thus, a different metric, such as Tchebycheff metric can be used in those cases. Nevertheless, the overall idea is that once NSGA-II finds a set of solutions close to the true Pareto-optimal region, we use a local search technique from each of these solutions with a differing emphasis of objective functions in the hope of better converging to the true Pareto-optimal front. Since independent local search methods are tried from each solution obtained using an MOEA, all optimized solutions obtained by the local search method need not be non-dominated to each other. Thus, we find the non-dominated set of solutions from the obtained set of solutions before proceeding further.

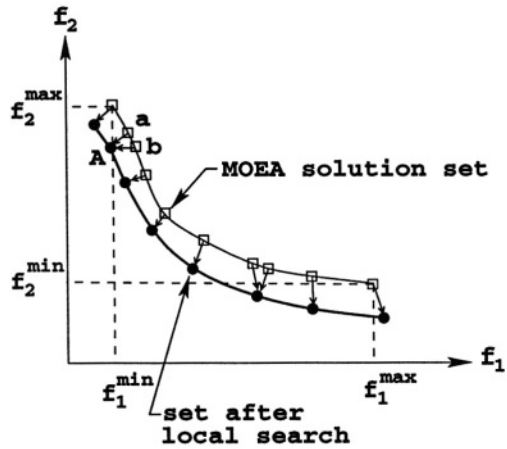


Figure 6.5. The local search technique is expected to find better solutions.

The complete procedure of the proposed hybrid strategy is shown in Figure 6.6. Starting from the MOEA results, we first apply a local search technique, followed by a non-domination check. After non-dominated

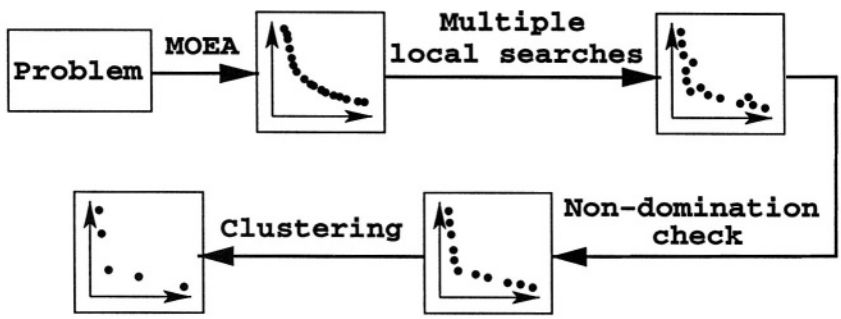


Figure 6.6. The proposed hybrid procedure of using a local search technique, a non-domination check, and a clustering technique are illustrated.

solutions are found, a clustering technique is used to reduce the size of the optimal set, as discussed in the next subsection.

4.2 Reducing the size of non-dominated set

In an ideal scenario, user is interested in finding a good spread of non-dominated solutions closer to the true Pareto-optimal front. From a practical standpoint, the user would be interested in a handful of solutions (in most cases, 5 to 10 solutions are probably enough). Interestingly, most MOEA studies use a population of size 100 or more, thereby finding about 100 different non-dominated solutions. The interesting question to ask is 'Why are MOEAs set to find many more solutions than desired?'

The answer is fundamental to the working of an EA. The population size required in an EA depends on a number of factors related to the number of decision variables, the complexity of the problem, and others (Goldberg et al, 1992; Harik et al., 1999). The population cannot be sized according to the desired number of non-dominated solutions in a problem. Since in most interesting problems, the number of decision variables are large and are complex, the population sizes used in solving those problems can be in hundreds. Such a population size is mandatory for the successful use of an EA. The irony is that when an MOEA works well with such a population size N, eventually it finds N different non-dominated solutions, particularly if an adequate niching mechanism used. good. Thus, we need to devise a separate procedure of selecting a handful of solutions from a rather large obtained set of non-dominated solutions.

One approach would be to use a clustering technique similar to that used in (Zitzler, 1999) for reducing the size of the obtained non-dominated set of solutions. In this technique, each of N solutions is assumed to belong to a separate cluster. Thereafter, the distance d_c between all pairs of clusters is calculated by first finding the centroid of each cluster and then calculating the Euclidean distance between the centroids. Two clusters having the minimum distance are merged together into a bigger cluster. This procedure is continued till the desired number of clusters are identified. Finally, with the remaining clusters, the solution closest to the centroid of the cluster is retained and all other solutions from each cluster are deleted. This is how the clusters can be merged and the cardinality of the solution set can be reduced. Figure 6.7 shows the MOEA solution set in open boxes and the reduced set in solid boxes. Care may be taken to choose the extreme solutions in the extreme clusters.

However, in many problems the local search strategy itself can reduce the cardinality of the obtained set of non-dominated solutions. This will particularly happen in problems with a discrete search space. For two closely located solutions, the pseudo-weight vectors may not be very

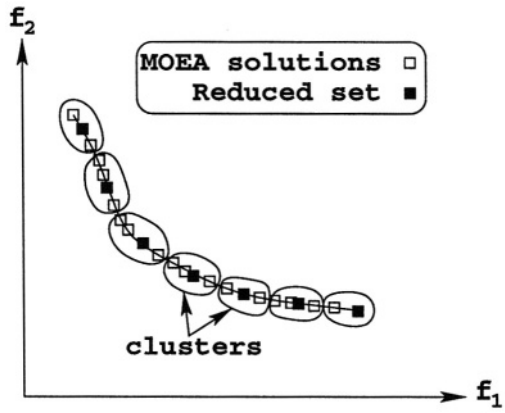


Figure 6.7. The clustering method of reducing the set of non-dominated solutions is illustrated.

different. Thus, when a local search procedure is started from each of these solutions (which are close to each other) with a $F(\mathbf{x})$ which is also similar, the resulting optimum solutions may be identical in a discrete search space problem. The solutions a and b in Figure 6.5 are close and after the local search procedure they may converge to the same solution A. Thus, for many solutions obtained using NSGA-II, the resulting optimum obtained using the local search method may be the same. Thus, the local search procedure itself may reduce the size of the obtained non-dominated solutions in problems with a finite search space. Figure 6.6 shows that clustering is the final operation of the proposed hybrid strategy.

5. Optimal Shape Design

Designing shape of engineering components is not a new activity. The use of optimization in engineering shape design has also received a lot of attention. However, in optimal shape design problems, the most popular approach has been to pre-define a parametric mathematical function for the boundary describing the shape and use an optimization technique to find the optimal values of the parameters describing the mathematical function. Although this procedure requires prior knowledge of the shape, the classical optimization methods facilitated the optimization of parameters describing the mathematical shape functions.

With the advent of evolutionary algorithms as an alternate optimization method, there exist a number of applications of optimal shape de-

sign, where shapes are evolved by deciding presence or absence of a number of small elements (Chapman and Jakiela, 1996; Chapman et al., 1994; Hamada and Schoenauer, 2000; Jakiela et al., 2000; Sandgren et al., 1990). A predefined area (or volume) is divided into a number of small regular elements. The task of an evolutionary optimization procedure is to find which elements should be kept and which should be thrown away so that the resulting shape is optimal with respect to an objective function. This procedure has a number of advantages:

1 The use of numerical finite element method (or boundary element method) is a usual method of analyzing an engineering component. Since the finite element procedure requires the component to be divided into a number of small elements, this approach reduces one computational step and is complimentary to the usual finite element method.

2 Since no apriori knowledge about the shape is required, this method does not have any bias from the user.

3 By simply using three-dimensional elements, the approach can be extended to three-dimensional shape design problems.

4 The number and shape of holes in a component can evolve naturally without explicitly fixing them by the user.

Most studies of this method, including the studies with evolutionary algorithms, have concentrated on optimizing a single objective. In this study, we apply this evolutionary procedure for multiple conflicting objectives.

5.1 Representation

In this study, we consider two-dimensional shape design problems only. However, the procedure can be easily extended to three-dimensional shape design problems. We begin with a rectangular plate, describing the maximum overall region, where the shapes will be confined. Thereafter, we divide the rectangular plate into a finite number of small elements (refer to Figure 6.8). We consider here square elements, although any other shape including triangular or rectangular elements can also be considered. Since the presence or absence of every element is a decision variable, we use a binary coding describing a shape. For the shape shown in Figure 6.9, the corresponding binary coding is as follows:

```
01110 11111 10001 11111
```

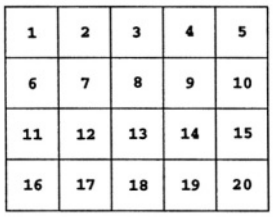


Figure 6.8. Rectangular plate is divided into small elements.

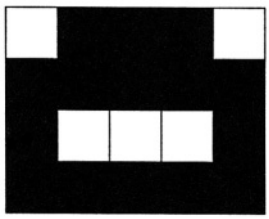


Figure 6.9. The skeleton of a shape is shown.

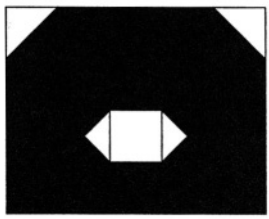


Figure 6.10. Final smoothened shape is shown.

The presence is denoted by a 1 and the absence is shown by a 0. A left-to-right coding procedure as shown in Figure 6.8 is adopted here. In order to smoothen the stair-case like shape denoted by the basic skeleton representation, we add triangular elements (shown shaded) for different cases in Figure 6.11. The resulting skeleton shape shown in Figure 6.9 represents the true shape shown in Figure 6.10.

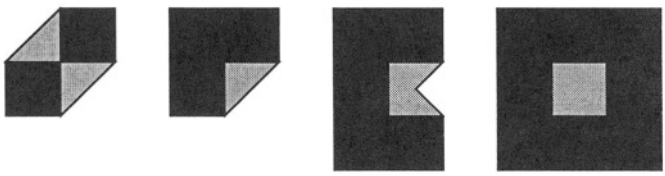


Figure 6.11. Different cases of smoothening through triangular elements are shown.

5.2 Evaluation

When the shape is smoothened, the shape is further divided into smaller elements. All interior rectangular elements are divided into two triangles and all boundary elements (including elements around a hole) are divided into four small triangles. Even the boundary triangles used for smoothening is divided into smaller triangles. The shape is evaluated by finding the maximum stress and deflection developed at any point in the component by the application of the specified loads. Since no connectivity check is made while creating a new string or while creating the initial random population, a string may represent a number of disconnected regions in the rectangle. In this case, we proceed with the

biggest cluster of connected elements (where two elements are defined to be connected if they have at least one common corner). The string is repaired by assigning a 0 at all elements which are not part of the biggest cluster.

In all applications here, two conflicting objectives are chosen: weight and deflection. These two objectives are conflicting because a minimum weight design is usually not stiff and produces a large deflection, whereas a minimum deflection design has densely packed elements, thereby causing a large weight of the overall component. The maximum stress and deflection values are restricted to lie within specified limits of the design by using them as constraints.

6. Simulation Results

To show the efficacy of the proposed hybrid multi-objective optimization procedure in solving optimal shape design problems, we use a number of mechanical component design problems. In all cases, we use NSGA-II as the multi-objective optimizer. Since binary-coded strings are used to represent a shape, we use a bit-wise hill-climbing strategy as the local search operator. The procedure is simple. Starting from the left of the string, every bit is flipped to see if it improves the design. If it does, the flipped bit is retained, else the bit is unchanged. This procedure is continued until no bit-flipping over the entire string length has resulted an improvement.

Since the shapes are represented in a two-dimensional grid, we introduce a new crossover operator which respects the rows or columns of two parents. Whether to swap rows or columns are decided with a probability 0.5. Each row or column is swapped with a probability $0.95/d$, where d *is* the number of rows or columns, as the case may be. This way on an average one row or one column will get swapped between the parents. A bit-wise mutation with a probability of 1/string-length is used. NSGA-II is run for 150 generations. It is important to highlight that NSGA-II does not require any extra parameter setting. In all problems, a population of size 30 is used.

For all problems, we use the following material properties:

Plate thickness	: 50 mm
Yield strength	: 150 MPa
Young's modulus	: 200 GPa
Poisson's ratio	: 0.25

6.1 Cantilever Plate Design

First, we consider a cantilever plate design problem, where an end load $P = 10$ kN is applied as shown in Figure 6.12. The rectangular plate of

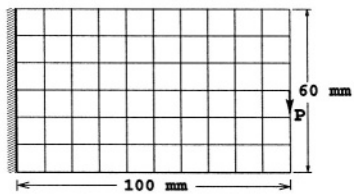


Figure 6.12. The loading and support of the cantilever plate are shown.

size 60×100 mm^2 is divided into 60 small rectangular elements. Thus, 60 bits are used construct a binary string representing a shape. Although symmetry information could have been used to reduce the number of decision variable to 30, we do not use any such information here to give NSGA-II a stringent test for investigating if a near-symmetric shape appears.

Figure 6.13 shows the four steps of the proposed hybrid method in designing the cantilever plate. The first plot shows the non-dominated solutions obtained using NSGA-II. Since the population size is 30, NSGA-II is able to find 30 different non-dominated solutions. Thereafter, the local search method is applied from each non-dominated solution and new and improved set of solutions are obtained. The third plot is the result of the non-dominated check of the solutions obtained after the local search method. Three dominated solutions are eliminated by this process. The final plot is obtained after the clustering operation with a choice of nine solutions. The plot shows how a set of nine well distributed solutions is found from the third plot of 27 solutions. If fewer than nine solutions are desired, the clustering mechanism can be set accordingly.

In order to visualize the obtained set of nine solutions having a wide range of trade-offs in the weight and scaled deflection values, we show the shapes in Figure 6.14. It is clear that starting from a low-weight solution (with large deflection), how large-weight (with small deflection) shapes are found by the hybrid method. It is interesting to note that the minimum weight solution eliminated one complete row (the bottom-most row) in order to reduce the overall weight. The second solution (the element (1,2) in the 3×3 matrix in Figure 6.14) corresponds to the second-best weight solution. It is well known that for an end load cantilever plate, a parabolic shape is optimal. Both shapes (elements

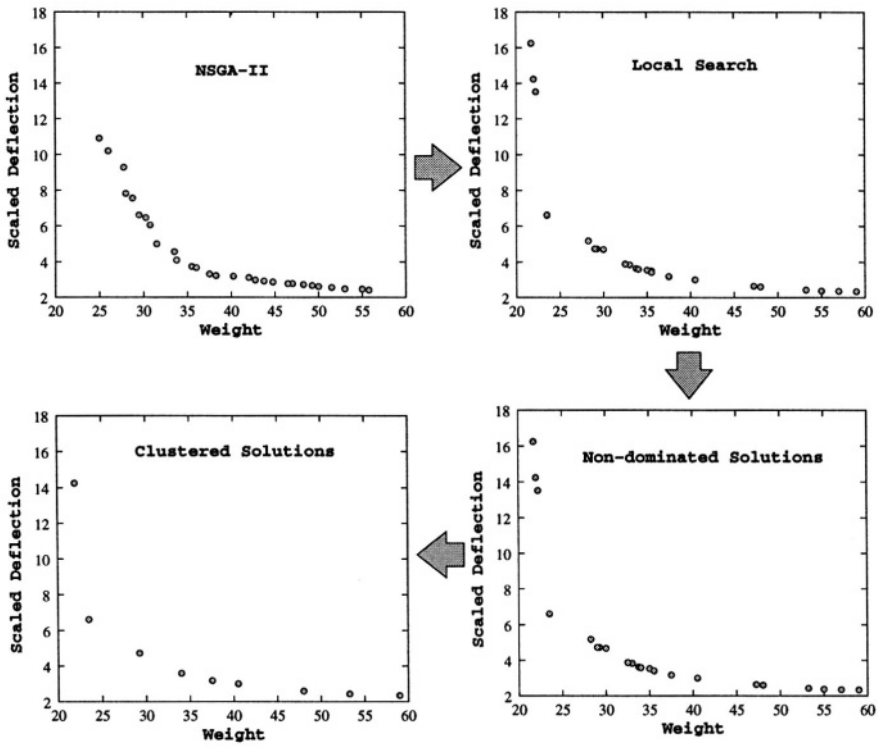


Figure 6.13. The steps of the hybrid procedure to find nine trade-off solutions for the cantilever plate design problem are shown.

(1,1) and (1,2)) exhibits a similar shape. As the importance of deflection increases, the shapes tend to have more and more elements, thereby making the plate rigid enough to have smaller deflection. In the middle, the development of vertical stiffener is interesting. This is a compromise between the minimum weight solution and a minimum deflection solution. By adding a stiffener the weight of the structure does not increase much, whereas the stiffness of plate increases (hence the deflection reduces). Finally, the complete plate with right top and bottom ends chopped off is the minimum deflection solution.

We would like to reiterate here that the above nine solutions are not results of multiple runs of a multi-objective optimization algorithm. All nine solutions (and if needed, more can also be obtained) with interesting trade-offs between weight and deflection are obtained in one simulation run of the hybrid method.

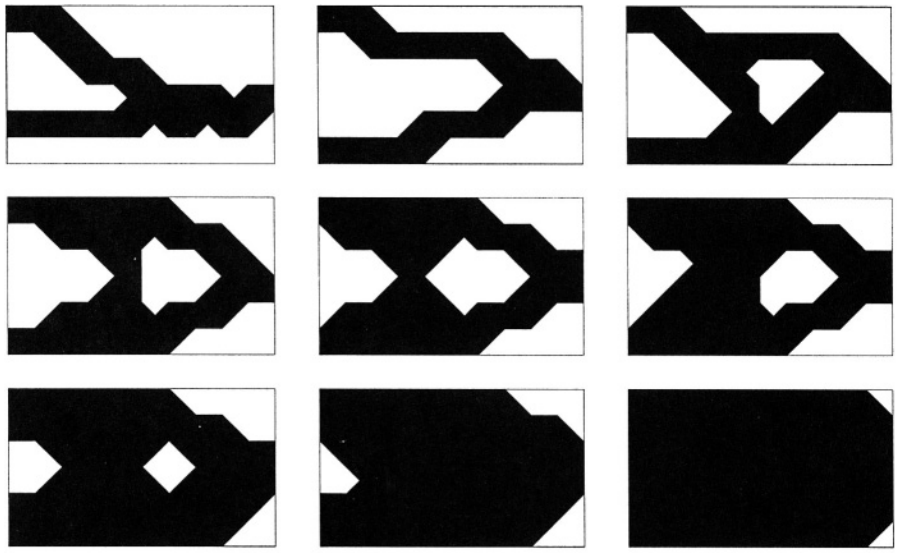


Figure 6.14. Nine trade-off shapes for the cantilever plate design are shown.

6.2 Simply-Supported Plate Design

Next, we consider a simply-supported plate design, starting from a rectangular plate of identical dimension as in the previous design. The plate is supported on two supports as shown in Figure 6.15 and a vertical load P = 10 kN is acted on the top-middle node of the plate.

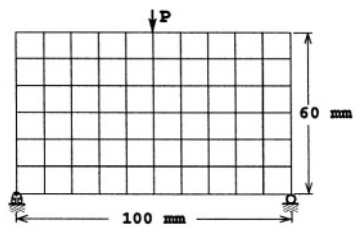


Figure 6.15. The loading and support of the simply-supported plate are shown.

Figure 6.16 shows the obtained non-dominated solutions using NSGA-II. After local search method, the obtained non-dominated solutions have a wider distribution. The number of solutions have been reduced from 30 solutions to 22 solutions by the non-dominated checking. Finally, the clustering algorithm finds nine widely separated solutions from 22 non-dominated solutions. The shape of these nine solutions are shown

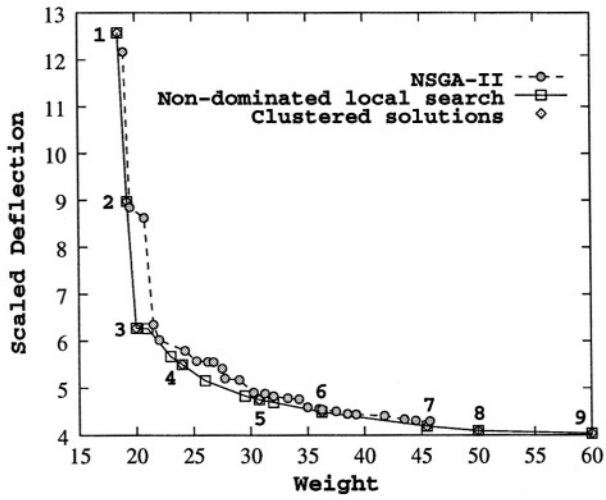


Figure 6.16. Hybrid procedure finds nine trade-off solutions for the simply-supported plate design problem.

in Figure 6.17. The minimum weight solution tends to use one row (the

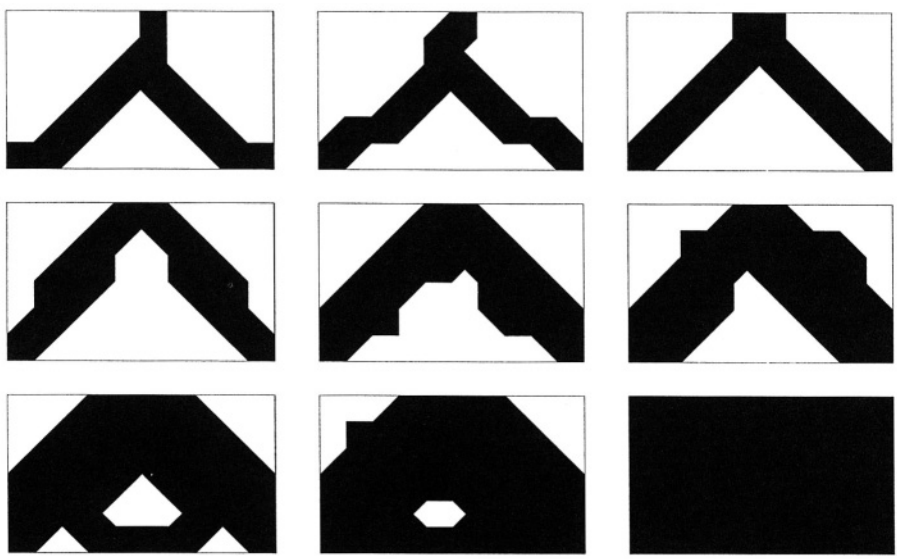


Figure 6.17. Nine trade-off shapes for the simply-supported plate design are shown.

top-most row) less, but since the load is acting on the top of the plate,

one element is added to have the load transferred to the plate. The third solution (shown in the (1,3)-th position in the matrix) is interesting. A careful look at Figure 6.16 reveals that this solution is a 'knee' solution. To achieve a small advantage in weight-loss, a large sacrifice in the deflection-gain is evident. Similarly, to achieve a small advantage in deflection-loss, a large sacrifice in weight is needed. Shapes in position (1,2) and (2,1) can be compared with respect to the shape in position (1,3). Shape in position (3,1) or solution 7 is also interesting. In order to have further reduction in deflection stiffening of the two slanted arms is needed. Finally, the absolute minimum deflection shape is the complete rectangle with maximum possible weight.

Starting with the minimum weight design having two slim slanted legs down to thickening the legs to make them stiff, followed by joining the legs with a stiffener, and finally finding the complete rectangular plate having minimum deflection are all intuitive trade-off solutions. In the absence of any such knowledge, it is interesting how the hybrid procedure with NSGA-II is able to find the whole family of different trade-off solutions.

6.3 Hoister Plate Design

Here, we attempt to design a hoister, that lifts a weight. Once again, a rectangular plate is used, but this time we use a 80×60 mm^2 plate with 48 elements. Thus, the string to represent a hoister is 48 bits long. The loading and the support conditions are shown in Figure 6.18. The vertical load is 5 kN and the horizontal loadings are 2.5 kN distributed over the length of the element. In order to apply the load, two center elements are forcibly made absent (that is, the corresponding bits in a string are always set to zero).

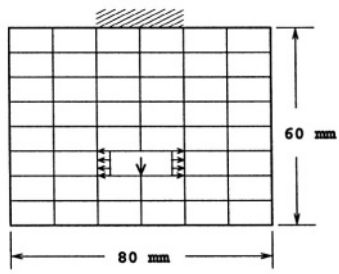


Figure 6.18. The loading and support of the hoister plate are shown.

Figure 6.19 shows 30 non-dominated solutions obtained using NSGA-II. The figure shows a good spread of trade-off solutions. When the local search method is applied from each solution and a non-domination check is made, there exists only 14 solutions, as shown in Figure 6.20. Moreover, most solutions concentrate near the middle of the previously-obtained front. Although two extreme solutions are found, the diversity among solutions is somewhat lost. Many solutions near the minimum

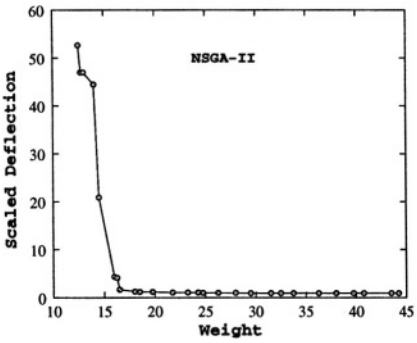


Figure 6.19. NSGA-II solutions for the hoister plate design problem are shown.

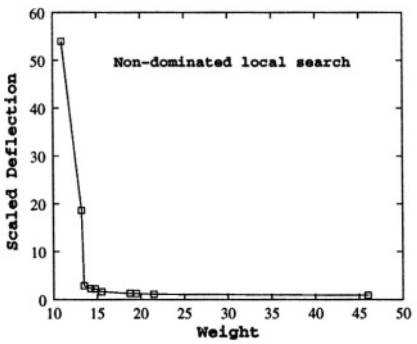


Figure 6.20. Solutions obtained after the local search method for the hoister plate design problem are shown.

deflection region in Figure 6.19 converge to an identical solution after the local search method. Although this allows the procedure to converge to a reduced set of non-dominated solutions, some important trade-off solutions may get lost by this process. This happens mainly due to the way we have allocated a pseudo-weight to a solution.

6.3.1 Continuously updated pseudo-weight vector. We have highlighted earlier that the pseudo-weight vector computation using equation 6.5 provides an approximate idea of the true weights associated with objectives at any solution. In order to get a better approximation of the true weight, we propose a continuously-updated pseudo-weight vector, where solutions are not assigned a fixed weight vector according to equation 6.5, instead the optimum found after each local search is used to update the weight vector.

Let us illustrate the procedure with the help of Figure 6.21 and for two objectives. For more than two objectives, a similar procedure can be adopted easily. First, the extreme solutions A and B are assigned a weight vector according to equation 6.5. A local search procedure is used

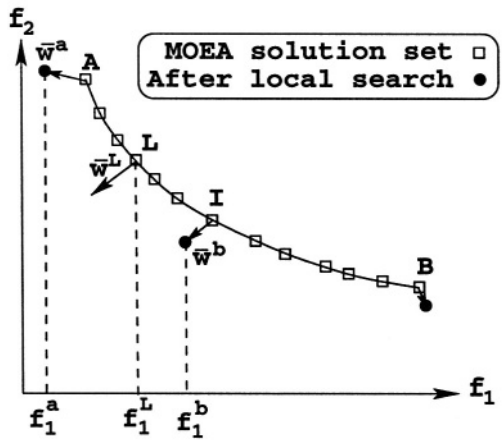


Figure 6.21. The continuously-updated weighted approach is illustrated.

to find the corresponding optimum solutions. Based on the new values of $f_j^{\min}$ and $f_j^{\max}$ values, an intermediate solution I is assigned a weight vector. Preferably, this intermediate solution I can be chosen as the one being maximally away from the extreme solutions. The local search procedure can be used with this intermediate solution and the optimum solution can be found. Now we consider each of the two intervals: region between A and I and the region between I and A. An intermediate solution (preferably the solution in the middle of the range) is chosen in each range. Say the solution L is chosen in the range between A and I. The pseudo-weight vector for this solution is calculated as follows.

$$\bar{w}_j^{\mathrm{L}} = \bar{w}_j^{\mathrm{a}} \frac{f_j^{\mathrm{b}} - f_j^{\mathrm{L}}}{f_j^{\mathrm{b}} - f_j^{\mathrm{a}}} + \bar{w}_j^{\mathrm{b}} \frac{f_j^{\mathrm{L}} - f_j^{\mathrm{a}}}{f_j^{\mathrm{b}} - f_j^{\mathrm{a}}}, \tag{6.6}$$

where solutions a and b are extreme solutions for the range under consideration. Thus, for solution L, solution a is A and solution b is I. The parameters $\bar{w}_j^{\mathrm{a}}$ and $\bar{w}_j^{\mathrm{b}}$ are pseudo-weights associated with the extreme solutions. When pseudo-weights are calculated for objective functions, they can be normalized. This procedure can be repeated till the local search is applied to all solutions. This update of pseudo-weights with a local search procedure is advantageous in two ways:

1 The true optimum solution corresponding to a weight vector can be found, and

2 The calculated weights for each optimum solution is close to their true weights.

Figure 6.22 shows the obtained non-dominated solutions using the hybrid method with the above continuously updated pseudo-weight vector calculation. The figure shows that a good distribution of solutions is

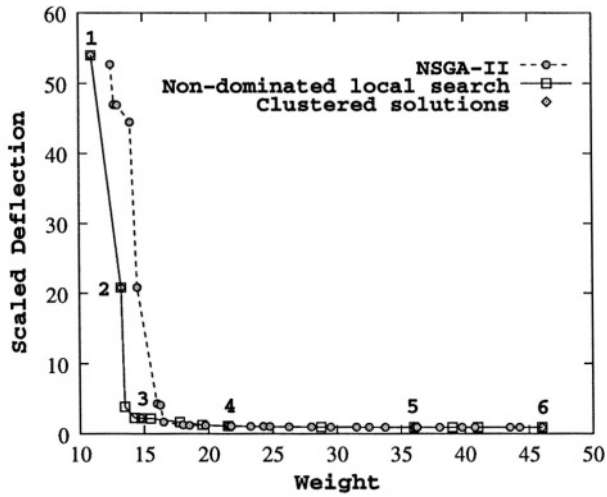


Figure 6.22. Hybrid procedure finds six trade-off solutions for the hoister plate design problem.

now possible to obtain with the continuously updated scheme. There are 17 solutions found after the local search method, but importantly they are well distributed.

During clustering, we choose to have only six solutions, which are marked in the figure. The figure shows the spread of obtained solutions. It is interesting to note here that solutions 4, 5 and 6 have very similar deflection, but their weights are very different. Figure 6.23 shows the shape of six trade-off solutions. The minimum weight solution is our familiar hook, often used for hoisting purposes. What is interesting is that the opening in the left side of the hook evolves without any predefined knowledge. The minimization of overall weight finds the hook as the optimum design. It is also interesting that this design does not use two columns (left-most and right-most) and one row (bottom-most), in order to minimize the weight. From the support in the middle of the plate, the shape turns to accommodate the loading nodes, just like a human designer would do in designing a hook. As more stiff designs are

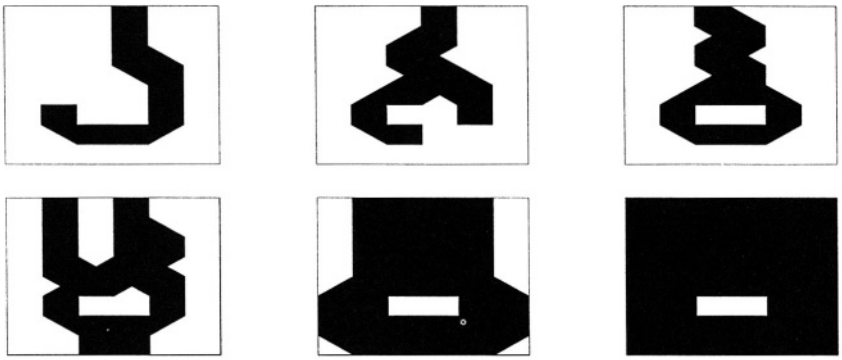


Figure 6.23. Six trade-off shapes for the hoister plate design are shown.

desired, the gap in the hook closes and in the third solution, the loop is closed. This way, a much stiffer hoister plate is designed. Other three designs are different ways to make the plate more stiff, but mainly by significantly increasing the weight for a small gain in the stiffness.

6.4 Bicycle Frame Design

Finally, we attempt to design a bicycle frame for a vertical load of 10 kN applied at A in Figure 6.24. The specifications are similar to that used elsewhere (Kim et al., 2000). The plate is 20 mm thick and is restricted to be designed within the area shown in Figure 6.24. The frame is supported at two places B and C. The point B marks the

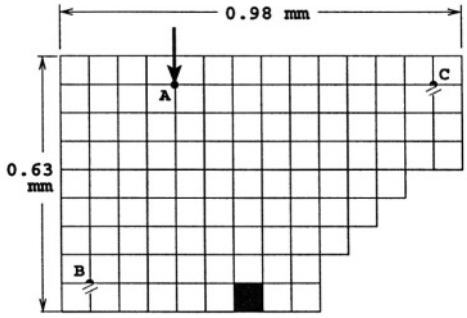


Figure 6.24. The hybrid procedure is illustrated for the bicycle frame design.

position of the axle of the rear wheel and the point C is the location of the handle support. The filled element is the location of the pedal

assembly and is always present. The material yield stress is 140 MPa, Young's modulus is 80 GPa and Poisson's ratio is 0.25. The maximum allowed displacement is 5 mm.

Figure 6.25 shows the NSGA-II solutions and corresponding solutions obtained by the hybrid approach. Here, we are interested in finding four different trade-off solutions, marked in the figure.

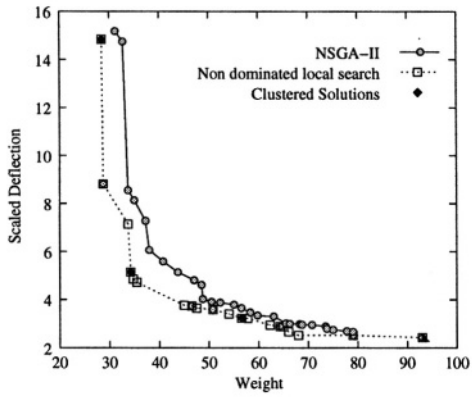


Figure 6.25. The hybrid procedure is illustrated for the bicycle frame design.

These four solutions are shown mounted on a sketch of a bicycle in Figure 6.26. The top-left solution is the minimum weight design. The second solution have put more material on the bar joining the seat and the rear wheel axle. Third solution joins the paddle wheel with the bar joining the handle and the seat. This solution is similar to the one seen on roads. Maximum weight solution puts almost full plate to have the most stiff bicycle frame. The interior hole and absence of top-left elements are all intuitive. The proposed hybrid approach can evolve such solutions without the knowledge and mainly by finding and maintaining trade-off solutions among weight and deflection. The presence of many such solutions with different trade-offs between weight and stiffness provides a plethora of information about various types of designs.

7. Conclusion

The hybrid multi-objective optimization technique proposed in this paper uses a combination of an multi-objective evolutionary algorithm (MOEA) and a local search operator. The proposed technique ensures

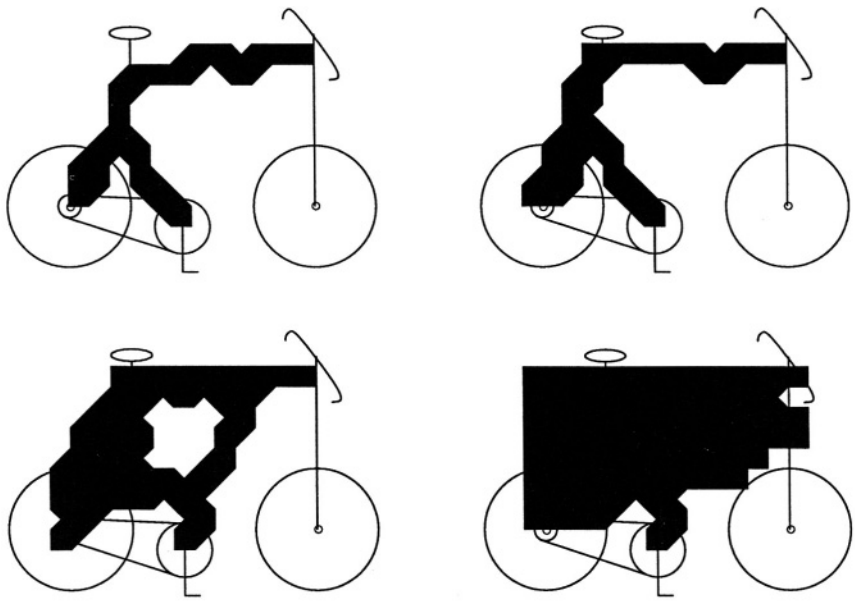


Figure 6.26. Four trade-off shapes for the bicycle frame design are shown.

a better convergence of MOEAs to the true Pareto-optimal region and helps in finding a small set of diverse solutions for practical reasons.

The efficacy of the proposed technique is demonstrated by solving a number of engineering shape design problems for two conflicting objectives —weight of the structure and maximum deflection of the structure. In all cases, the proposed technique has been shown to find a set of four to nine diverse solutions better converged than an MOEA alone. The results are encouraging and take the evolutionary multi-objective optimization approach much closer to practice.

Acknowledgments

The authors wish to thank Ministry of Human Resources Development (MHRD) for supporting this research.

References

Chankong, V. and Haimes, Y. Y. (1983). *Multiobjective decision making theory and methodology.* North-Holland, New York.

Goldberg, D. E., Deb, K., and Clark, J. H. (1992). Genetic algorithms, noise, and the sizing of populations. *Complex Systems,* 6:333–362.

Chapman, C. D. and Jakiela, M. J. (1996). Genetic algorithms based structural topology design with compliance and topology simplification considerations. *ASME Journal of Mechanical Design,* 118:89–98.

Kim, H., Querin, O. M., and Steven, G. P. (2000). Post-processing of the two-dimensional evolutionary structure optimization topologies. *In Evolutionati Design and Manufacture, London: Springer,* 33–44.

Chapman, C. D., Saitou, K., and Jakiela, M. J. (1994). Genetic algorithms as an approach to configuration and topology design. *ASME Journal of Mechanical Design,* 116:1005–1012.

Deb, K. (2001). *Multiobjective optimization using evolutionary algorithms.* Wiley, Chichester.

Deb, K. and Agrawal, R. B. (1995). Simulated binary crossover for continuous search space. *Complex Systems,* 9:115–148.

Deb, K., Agrawal, S., Pratap, A., and Meyarivan, T. (2000a). A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: Nsga-ii. In *Proceedings of the Parallel Problem Solving from Nature VI Conference,* pages 849–858.

Deb, K., Pratap, A., Agrawal, S., and Meyarivan, T. (2000b). *A fast and elitist multi-objective genetic algorithm: NSGA-II.* Technical Report No. 2000001, Indian Institute of Technology Kanpur, India.

Duda, J. W. and Jakiela, M. J. (1997). Generation and classification of structural topologies with genetic algorithm speciation. *ASME Journal of Mechanical Design,* 119:127–131.

Fonseca, C. M. and Fleming, P. J. (1993). Genetic algorithms for multiobjective optimization: Formulation, discussion, and generalization. In *Proceedings of the Fifth International Conference on Genetic Algorithms,* pages 416–423.

Hamada, H. and Schoenauer, M. (2000). Adaptive techniques for evolutionary optimum design. In *Proceedings of the Evolutionary Design and Manufacture,* pages 123–136.

Harik, G., Cantu-paz, E., Goldberg, D. E., and Miller, B. L. (1999). The gambler's ruin problem, genetic algorithms, and the sizing of populations. *Evolutionary Computation,* 7(3):231–254.

Horn, J., Nafploitis, N., and Goldberg, D. E. (1994). A niched pareto genetic algorithm for multi-objective optimization. In *Proceedings of the First IEEE Conference on Evolutionary Computation,* pages 82–87.

Jakiela, M. J., Chapman, C., Duda, J., Adewuya, A., and abd Saitou, K. (2000). Continuum structural topology design with genetic algorithms. *Computer Methods in Applied Mechanics and Engineering,* 186:339–356.

Knowles, J. and Corne, D. (1999). The pareto archived evolution strategy: A new baseline algorithm for multiobjective optimisation. In *Proceedings of the 1999 Congress on Evolutionary Computation, Piscataway: New Jersey: IEEE Service Center,* pages 98–105.

Miettinen, K. (1999). *Nonlinear multiobjective optimization.* Kluwer, Boston.

Parmee, I. C., Cvetkovic, D., Watson, A. H., and Bonham, C. R. (2000). Multiobjective satisfaction within an interactive evolutionary design environment. *Evolutionary Computation,* 8(2):197-222.

Sandgren, E., Jensen, E., and Welton, J. (1990). Topological design of structural components using genetic optimization methods. In *Proceedings of the Winter Annual Meeting of the American Society of Mechanical Engineers,* pages 31–43.

Sen, P. and Yang, J. B. (1998). *Multiple criteria decision support in engineering design.* Springer, London.

Srinivas, N. and Deb, K. (1995). Multi-objective function optimization using non-dominated sorting genetic algorithms. *Evolutionary Computation,* 2:221–248.

Steuer, R. E. (1986). *Multiple criteria optimization: Theory, computation, and application.* Wiley, New York.

Zitzler, E. (1999). *Evolutionary algorithms for multiobjective optimization: Methods and applications.* Doctoral thesis ETH NO. 13398, Zurich: Swiss Federal Institute of Technology (ETH), Aachen, Germany: Shaker Verlag.

Zitzler, E. and Thiele, L. (1998). *An evolutionary algorithm for multiobjective optimization: The strength Pareto approach.* Technical Report No. 43 (May 1998), Zürich: Computer Engineering and Networks Laboratory, Switzerland.

This page intentionally left blank

Chapter 7

ASSESSMENT METHODOLOGIES FOR MULTIOBJECTIVE EVOLUTIONARY ALGORITHMS

Ruhul Sarker and
Carlos A. Coello Coello

Abstract The Pareto-based evolutionary multiobjective algorithms have shown some success in solving multiobjective optimization problems. However, it is difficult to judge the performance of multiobjective algorithms because there is no universally accepted definition of *optimum* in multiobjective as in single-objective optimization problems. As appeared in the literature, there are several methods to compare two or more multiobjective algorithms. In this chapter, we discuss the existing comparison methods with their strengths and weaknesses.

Keywords: multiobjective, Pareto front, metrics, evolutionary algorithm, performance assessment

1. Introduction

In multiobjective optimization problems, we define the vector-valued objective function $\mathbf{f} : R^n \rightarrow R^m$ where $m > 1$, n is the dimension of the decision vector, $\mathbf{x}$, and m is the dimension of the objective vector, $\mathbf{y}$. For the case of minimization problems, we intend to minimize

$$\mathbf{y} = \mathbf{f}(\mathbf{x}) = (f_1(\mathbf{x}), \cdots, f_m(\mathbf{x})),$$

where $\mathbf{x} \in R^n$, and $\mathbf{y} \in R^m$. A solution $\mathbf{y_a}$ is said to dominate solution $\mathbf{y_b}$ if $y_{ai} \leq y_{bi}, \forall i \in \{1, \cdots, m\}$ and $y_{ai} < y_{bi}, \exists i \in \{1, \cdots, m\}$. In most cases, the objective functions are in conflict, because in order to decrease any of the objective functions, we need to increase other objective functions. When we have a solution which is not dominated by any other solution in the feasible space, we call it Pareto-optimal.

The set of all Pareto-optimal solutions is termed the Pareto-optimal set, efficient set or admissible set. Their corresponding objective vectors are called the nondominated set.

Fonseca and Fleming (1995), categorized the evolutionary based techniques for multiobjective optimization into three approaches: plain aggregating approaches, population-based non-Pareto approaches, and Pareto-based approaches. A comprehensive survey of the different techniques under these three categories can be found in Coello (1999), in Veldhuizen (1999) and in the last two chapters of this book.

In multiobjective optimization, determining the quality of solutions produced by different evolutionary algorithms is a serious problem for researchers because there is no well accepted procedure in the literature. The problem is certainly difficult due to the fact that, unlike single-objective optimization, in this case we need to compare vectors (representing sets of solutions).

The chapter is organized as follows. We will start by surveying the most important work on performance assessment methodologies reported in the specialized literature, describing each proposal and analyzing some of their drawbacks. Then, we will present a brief comparative study of the main metrics discussed and we will conclude with some possible paths of future research in this area.

2. Assessment Methodologies

The definition of reliable assessment methodologies is very important to be able to validate an algorithm. However, when dealing with multiobjective optimization problems, there are several reasons why the assessment of results becomes difficult. The initial problem is that we will be generating several solutions, instead of only one (we aim to generate as many elements as possible of the Pareto optimal set). The second problem is that the stochastic nature of evolutionary algorithms makes it necessary to perform several runs to assess their performance. Thus, our results have to be validated using statistical analysis tools. Finally, we may be interested in measuring different things. For example, we may be interested in having a robust algorithm that approximates the global Pareto front of a problem consistently, rather than an algorithm that converges to the global Pareto front but only occasionally. Also, we may be interested in analyzing the behavior of an evolutionary algorithm during the evolutionary process, trying to establish its capabilities to keep diversity and to progressively converge to a set of solutions close to the global Pareto front of a problem.

The previous discussion is a clear indication that the design of metrics for multiobjective optimization is not easy. The next topic to discuss is what to measure. It is very important to establish what sort of results we expect to measure from an algorithm so that we can define appropriate metrics.

Three are normally the issues to take into consideration to design a good metric in this domain (Zitzler et al., 2000):

1 Minimize the distance of the Pareto front produced by our algorithm with respect to the global Pareto front (assuming we know its location).

2 Maximize the spread of solutions found, so that we can have a distribution of vectors as smooth and uniform as possible.

3 Maximize the number of elements of the Pareto optimal set found.

Next, we will review some of the main proposals reported in the literature that attempt to capture these three issues indicated above. As we will see, none of these proposals really captures the three issues in a single value. In fact, to attempt to produce a single metric that captures these three issues may prove fruitless, since each of these issues refers to different performance aspects of an algorithm. Therefore, their fusion into a single value may be misleading. Interestingly, the issue of designing a good assessment measure for this domain is also a multiobjective optimization problem. The techniques that have attempted to produce a single value to measure the three issues previously discussed, are really some form of an aggregating function and therefore the problems associated with them (Coello, 1999). It is therefore, more appropriate to use different metrics to evaluate different performance aspects of an algorithm.

2.1 A Short Survey of Metrics

As we will see in our following survey of metrics, most of the current proposals assume that the global Pareto front (PF_{global}) of the multiobjective optimization under study is known or it can be generated (e.g., through an enumerative approach).

If that is the case, we can test the performance of a multiobjective evolutionary algorithm (MEA) by comparing the Pareto fronts produced by our algorithm (PF_{obtain}) against the global Pareto front (PF_{global}[1]) and then determine certain error measures that indicate how effective is

[1]We have adopted a notation similar to the one proposed by Veldhuizen (Veldhuizen, 1999).

the algorithm analyzed. The error rate and generational distance metrics discussed next make this assumption.

2.1.1 Error Rate. This metric was proposed by Veldhuizen (1999) to indicate the percentage of solutions (from PF_{obtain}) that are not members of PF_{global}:

$$ER = \frac{\sum_{i=1}^{n} e_i}{n}, \tag{7.1}$$

where n is the number of vectors in PF_{obtain}; $e_i = 0$ if vector i is a member of PF_{global}, and $e_i = 1$ otherwise. It should then be clear that $ER = 0$ indicates an ideal behavior, since it would mean that all the vectors generated by our MEA belong to PF_{global}. Note that this metric requires knowing the number of elements of the global Pareto optimal set, which is often impossible to determine. Veldhuizen and Lamont (1998,1999) have used parallel processing techniques to enumerate the entire intrinsic search space of an evolutionary algorithm, so that PF_{global} can be found. However, they only reported success with strings of ≤ 26 bits. To enumerate the entire intrinsic search space of real-world multiobjective optimization problems with high dimensionality is way beyond the processing capabilities of any computer (otherwise, any multiobjective optimization problem could be solved by enumeration).

This metric clearly addresses the third of the issues discussed in Section 2.

2.1.2 Generational Distance. The concept of generational distance (GD) was introduced by Veldhuizen and Lamont (1998) as a way of estimating how far are the elements in PF_{obtain} from those in PF_{global} and is defined as:

$$GD = \frac{\sqrt{\sum_{i=1}^{n} d_i^2}}{n} \tag{7.2}$$

where n is the number of vectors in PF_{obtain} and d_i is the Euclidean distance (measured in objective space) between each of these and the nearest member of PF_{global}. It should be clear that a value of $GD = 0$ indicates that all the elements generated are in PF_{global}. Therefore, any other value will indicate how "far" we are from the global Pareto front of our problem. Similar metrics were proposed by Rudolph (1998), Schott (1995), and Zitzler et al. (2000).

This metric also refers to the third issue discussed in Section 2

2.1.3 Spread. The spread measuring techniques measure the distribution of individuals in PF_{obtain} over the nondominated region. For example, Srinivas and Deb (1994) proposed the use a chi-squared distribution:

$$SP = \sqrt{\sum_{i=1}^{q+1} \left(\frac{n_i - \bar{n}_i}{\sigma_i} \right)^2} \tag{7.3}$$

where: q is the number of desired (Pareto) optimal points (it is assumed that the $(q+1)$-th subregion is dominated by the q-th subregion), n_i is the number of individuals in the i-th niche (or subregion (Deb and Goldberg, 1989)) of the nondominated region, $\bar{n}_i$ is the expected number of individuals present in the i-th niche, and σ_i^2 is the variance of the individuals present in the i-th subregion of the nondominated region. Deb (1989) had previously used probability theory to estimate that:

$$\sigma_i{}^2 = \bar{n}_i(1 - \frac{\bar{n}_i}{P}), i = 1, 2, ..., q, \tag{7.4}$$

where P is the population size. Since the $(q + 1)$-th subregion is a dominated region, then $\bar{n}_{q+1} = 0$ (i.e., we do not want to have any individuals in that region). Also, Deb's study showed that:

$$\sigma_{q+1}^2 = \sum_{i=1}^{q} \sigma_i^2 \tag{7.5}$$

Then, if SP=0, it means that our MEA has achieved an ideal distribution of points. Therefore, low values of SP imply a good distribution capacity of a given MEA.

To analyze the distribution using this measure, the nondominated region is divided into a certain number of equal subregions (this number is given by the decision maker). Knowing the population size used by the MEA under study, we can compute the amount of expected individuals in each subregion. This value is then used to compute the deviation measure indicated above.

A similar metric, called "efficient set spacing" (ESS) was proposed by Schott (1995):

$$ESS = \sqrt{\frac{1}{e-1} \sum_{i=1}^{e} (\bar{d} - d_i)^2} \tag{7.6}$$

where:

$$d_i = \min_j \left\{ |f_1^i - f_1^j| + |f_2^i - f_2^j| \right\} \tag{7.7}$$

where: $j = 1, \ldots, e$, and $\bar{d}$ refers to the mean of all d_i and e is the number of elements of the Pareto optimal set found so far. If $ESS = 0$, then it means that our MEA is giving us an ideal distribution of the elements of our nondominated vectors.

Schott's approach is based on a Holder metric of degree one (Horn and Nafpliotis, 1993) and measures the variance of the distance of each member of the Pareto optimal set (found so far) with respect to its closest neighbor. Note, however, that, as indicated by Veldhuizen (1999), this metric has to be adapted to consider certain special cases (e.g., disjoint Pareto fronts), and it may also be misleading unless it is combined with a metric that indicates the number of elements of the global Pareto optimal set found (e.g., if we produce only two solutions, this metric would indicate an ideal distribution).

Although none of these spread metrics really requires that we know PF_{global}, they both implicitly assume that our MEA has converged to global nondominated solutions. Otherwise, knowing that our algorithm produces a good distribution of solutions may become useless.

It should be clear that these metrics refer to the second issue discussed in Section 2

2.1.4 Space Covered. Zitzler and Thiele (1999) proposed a similar metric called "Size of the Space Covered" (SSC). This metric estimates the size of the global dominated set in objective space. The main idea behind this metric is to compute the area of objective function space covered by the nondominated vectors that our MEA has generated. For biobjective problems, each dominated vector represents a rectangle defined by the points (0,0) and $(f_1(x_i), f_2(x_i))$, where $f_1(x_i)$ and $f_2(x_i)$ are nondominated solutions. Therefore, SSC is computed as the union of the areas of all the rectangles that correspond to the nondominated vectors generated. However, the main drawback of this metric is that it may be misleading when the shape of the Pareto front is non-convex (Zitzler and Thiele, 1999).

Laumanns et al. (1999), used the concept of space coverage for comparing problems with more than two objectives. In their paper, they considered an m-dimensional cuboid as a reference set, from which the MEA under investigation should cover the maximum possible dominated space. Every nondominated solution gives a cone of dominated solutions. The intersection of this cone with the reference cuboid (which is also a

cuboid) adds to the dominated volume. In calculating the dominated volume, the overlapping parts of different solutions are not counted multiple times. In this method, the reference cuboid is developed using the single objective optimal solutions. That means the single objective optimal solution must be either known or computed before judging the quality of a MEA. If the single objective solutions are not known, for a problem with m functions, one has to solve m single objective problems to generate the reference cuboid. Potentially, this could be a expensive process (computationally speaking), particularly when dealing with real-world applications.

The value of space coverage varies with the number of nondominated solution points and their distribution over the Pareto front. We can then see that this metric tries to combine into a single value the three issues discussed in Section 2 Therefore, as indicated by Zitzler et al. (2000) the metric will be ineffective in those cases in which two algorithms differ in more than one of these three criteria previously mentioned (i.e., distance, spread and number of elements of the global Pareto optimal set found). That is why Veldhuizen (1999) suggested to use a ratio instead, but his metric also assumes knowledge of PF_{global}.

2.1.5 Coverage Metric. Zitzler and Thiele (1999) proposed another metric for comparing the performances of different MEAs. In this case, two nondominated sets are compared calculating the fraction of each of them that is "covered" (or dominated) by the other one. Suppose there are two algorithms $A1$ and $A2$ to compare their performances. In this method, the resulting set of nondominated points from a single run of algorithm $A1$ and another from a single run of algorithm $A2$ are processed to yield two numbers: the percentage of points from algorithm $A1$ which are equal to or dominated by points from $A2$, and vice versa. Statistical tests can be performed on the numbers yielded from several such pairwise comparisons.

Let $X', X'' \subseteq X$ be two sets of decision vectors. The function CM maps the ordered pair (X', X'') to the interval [0,1] (Zitzler and Thiele, 1999):

$$CM(X', X'') = \frac{|\{a'' \in X''; a' \in X' : a' \succeq a''\}|}{|X''|}$$

If $CM(X', X'') = 1$, then it means that all points in X'' are dominated by or are equal to points in X'. If $CM(X', X'') = 0$, then it means that none of the points in X'' are covered by the set X'.

This method can be used to show whether the outcomes of one algorithm dominate the outcomes of another algorithm without indicating

how much better it is. Also, note that this comparison technique does not check the uniformity of solution points along the trade-off surface.

Another problem with this approach is that it may return a better value for an algorithm that produces a single vector, closer to the global Pareto optimal front than for another algorithm that produces several well-distributed vectors that are, however, farther from the global Pareto optimal front (Knowles and Corne, 2000).

This metric only refers to the first issue discussed in Section 2 In fact, this metric is meant to complement the Space Covered metric previously discussed (Zitzler and Thiele, 1999). In his dissertation, Zitzler (1999) proposed another metric called "Coverage difference of two sets", that solves some of the problems of the Coverage Metric.

2.1.6 Statistical Comparison. MEAs are usually run several times to end up with a set of alternative solutions for MOPs from which the decision maker has to choose one. So, when we compare different MEAs, it is logical to compare them statistically instead of comparing only scalar values as in the single objective case.

An statistical comparison method called "attainment surfaces" was introduced by Fonseca and Fleming (1996). Consider the approximations to the Pareto optimal front for two algorithms $A1$ and $A2$ as shown in Figure 7.1. The lines joining the points (solid for $A1$ and dashed for A2) indicate the so-called attainment surfaces. An attainment surface divides objective space into two regions: one containing vectors which are dominated by the results produced by the algorithm, and another one that contains vectors that dominate the results produced by the algorithm. As shown in Figure 7.1, a number of sampling lines ($L1$, $L2$,) can be drawn from the origin, which intersect the attainment surfaces, across the full range of the Pareto front (trade-off surfaces obtained). The range of the Pareto surface is usually specified by the decision maker. For a given sampling line, the intersection of an algorithm closer to the origin (assuming minimization for both objectives) is the winner. In our case, algorithm $A1$ is winner for line $L3$ and $A2$ is winner for line $L4$. Fonseca and Fleming's idea was to consider a collection of sampling lines which intersect the attainment surfaces across the full range of the Pareto frontier.

If MEAs are run r times, each algorithm will return r attainment surfaces, one from each run. Having these r attainment surfaces, some from algorithm $A1$ and some from algorithm $A2$, a single sampling line yields r points of intersection, one for each surface. These intersections form a univariate distribution, and therefore, we can perform standard non-parametric statistical procedures to determine whether or not the

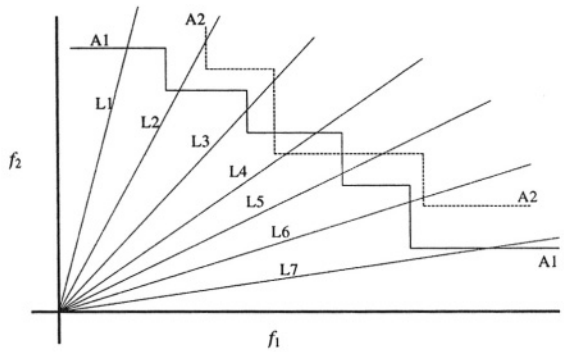


Figure 7.1. Sampling the Pareto frontier using lines of intersection

intersections for one of the algorithms occurs closer to the origin with some statistical significance. Such statistical tests have been performed by (Knowles and Corne, 2000) for each of several lines covering the Pareto tradeoff area. Insofar as the lines provide a uniform sampling of the Pareto surface, the result of this analysis yields two numbers—a percentage of the surface in which algorithm $A1$ outperforms algorithm A2 with statistical significance, and that when algorithm $A2$ outperforms algorithm $A1$.

Knowles and Corne (2000) presented their results of a comparison in the form of a pair $[a1,a2]$, where $a1$ gives the percentage of the space (i.e. the percentage of lines) on which algorithm $A1$ was found statistically superior to $A2$, and $a2$ gives the similar percentage for algorithm $A2$. Typically, if both $A1$ and $A2$ are 'good', then $a1 + a2 \¡ 100$. The quantity $[100 - (a1 + a2)]$, of course, gives the percentage of the space on which the results were statistically inconclusive. They use statistical significance at the 95 percent confidence level. Knowles and Corne (2000) also extended their comparison methodology for comparing more than two algorithms.

If the algorithms are competitive, the results may vary with the number of sampling lines drawn since the procedure considers only the intersection points of sampling lines and attainment surfaces. Knowles and Corne (2000) proposed that 100 lines must be adequate, although, obviously, more lines the better. They have shown experimentally that

the percentage of the space $(a1 + a2)$ increases, to give statistically significant results, with the increased number of lines.

A drawback of this approach is that it remains unclear how much better is one algorithm from another one against which it is being compared (Zitzler, 1999). An important aspect of this metric, however, is that the global Pareto optimal set does not need to be known.

Note that this metric is similar to the Coverage Metric, since they both compare the outcomes of two algorithms but none allows to determine how much better is an algorithm compared to others. This metric is also addressing only the first of the issues discussed in Section 2

In more recent work, da Fonseca et al. (2001) extended their original proposal relating their attainment function definition to random closed set theory (Matheron, 1975). Under this scheme, the attainment function can be seen as a generalization of the multivariate cumulative distribution function. The metric, in fact, addresses the three issues discussed in Section 2 However, its main drawback is that computing the multivariate empirical cumulative distribution function may be quite difficult and very expensive (computationally speaking) (da Fonseca et al., 2001).

3. Discussion

In practice, PF_{global} is a collection of points on the global Pareto front. To make metrics such as the Error Rate (*ER*) and Generational Distance (*GD*) meaningful, we must have a large number of points without any gap so that the global Pareto front looks like a continuous line instead of a dotted line.

Suppose that we have two algorithms ($A1$ and $A2$) to compare with respect to PF_{global}. Algorithm $A1$ produces only a few alternative solutions and all of these solutions are scattered and are members of PF_{global}, then *ER* (and *GD*) being zero indicates that the algorithm produces perfect results. Algorithm $A2$ produces many alternative but evenly distributed solutions. The *ER* (same for *GD*) value of $A2$ is greater than zero since there are few points who are not members of PF_{global}. Should we say $A1$ is better than $A2$? In a multiobjective decision process, more alternative solutions mean more flexibility to the human decision maker. Though the *Error Rate* and *Generational Distance* for $A1$ are zero, the number of alternative solutions and their distributions are not favorable for a good decision making process. So, it should be clear that *ER* and *GD* are not sufficient to judge the performance of MEAs.

The *Error Rate* and *Generational Distance* can be used, when the Pareto fronts are known, to get an indication of errors involved with the

obtained front. However, for an unsolved problem, it is impossible to know its Pareto front beforehand.

The *Spread* metric may help to judge an algorithm when the global Pareto front is unknown (although, this metric implicitly assumes that convergence to the global Pareto front has been achieved). However, it only judges the uniformity of solution points over different subregions without indicating the overall quality of solutions.

The *Space Covered* metric shows the dominated space covered by the solutions obtained from an algorithm. This metric, however, may be misleading when the Pareto front of the problem is concave.

The *Coverage Metric* shows whether the outcomes of one algorithm dominate the outcomes of another algorithm without indicating how much better it is. This comparison technique does not check the uniformity of solution points along the Pareto trade-off surface. Therefore, this metric will be misleading when algorithm $A1$ produces only one vector that is closer to the global Pareto front of the problem than the solutions produced by algorithm $A2$, which are uniformly distributed but are dominated by the solutions of $A1$.

The *Statistical Comparison* technique seems to be reasonably good in comparing different MEAs though the results may vary with the number of sampling lines. In contrast to other methods, this method is judging the overall quality of Pareto trade-off surface based on the sample information (taking only points on the intersection of attainment surfaces and sampling lines). In addition it does not explicitly measure the uniformity of the solutions points. Nevertheless, this metric is also incapable of establishing how much better is a certain algorithm with respect to others.

As discussed earlier, decision makers are interested in a good number of uniformly distributed solution points on the trade-off surface along their relative optimality. By the term *relative optimality* for a two objective minimization problem, we mean, the solution points (when plotted one objective versus the other) must be as close to the origin (0,0) as possible in a given subregion. No comparison technique gives a clear indication about the relative performance of an algorithm. However a combination of a few techniques and some other simple considerations may help to compare two or more algorithms up to a certain level.

The first thing is to plot the Pareto fronts and inspect visually their relative locations, relative number of points in each front and their distribution. The visual inspection is acceptable when one algorithm clearly outperforms the other. This method is limited to two or three function problems only. When the algorithms are very competitive, the statistical comparison along with the space covered, number of solution points,

average distance between neighboring points (with their distribution), and average distance between the origin (or any other reference point) and the solution points (with their distribution) may help to compare two or more MEAs.

There are other proposals for performance assessment measures in the specialized literature. For example, Zitzler et al. (2000) proposed a family of scale-dependent measures. Esbensen and Kuh (1996) and Ang et al. (2001) proposed the use of a utility function to determine the quality of the solutions produced by a MEA. This metric considers only nearness to the global Pareto optimal front, but not spread or number of elements achieved. Veldhuizen (1999) proposed a multiobjective version of the "progress measure" metric proposed by Thomas Bäck (1996). In this case, the relative convergence improvement of the MEA is measured. The main drawback of this metric is that it may be misleading in certain situations (e.g., when we find a single vector that belongs to a Pareto front closer to the global Pareto front than those vectors previously found by the algorithm).

However, it is important to keep in mind that combining the three issues discussed before is still an open research area, and it currently seems better to apply independently more than one metric to the same problem and analyze the combination of results found.

Notice also that some specific domains may allow the definition of more precise metrics. Such is the case of multiobjective combinatorial optimization, in which several quality measures have been proposed (see for example (Jaszkiewicz et al., 2001; Carlyle et al., 2001)).

4. Comparing Two Algorithms: An Example

In evolutionary computation, when comparing two algorithms, it is considered common practice that they both have the same representation (e.g., binary), operators (i.e., crossover and mutation), and parameters (i.e., crossover and mutation rates, population size, etc.) on all the functions to be tested. Otherwise, the comparison is not considered fair. For example, the comparison between an algorithm AL_1 with binary representation (with one-point crossover and bit-flip mutation) and another algorithm AL_2 with real number representation (with Gaussian mutation) would be treated as an improper comparison even if they both use the same parameters. If different parameters are used (e.g., different population sizes), it may become obvious to anyone that the amount of fitness function evaluations of the algorithms compared will be different and, therefore, any comparison will be unfair. However, if only the representation and operators are different, it is perhaps not so obvious why

the comparison is unfair. The reasons for that assumption are related to the role of the representation and the genetic operators in an evolutionary algorithm (Whitley et al., 1998; Caruana and Schaffer, 1988; Ronald, 1997; Eshelman et al., 1989; Spears and Jong, 1991; Fogarty, 1989).

If the representation, operators of two algorithms are same, the superiority of one algorithm over the other really shows the strength of the algorithmic design for the given representation and operators. We must mention here that there is a lot of empirical evidence regarding the fact that the real number representation works better than binary representation for most real valued problems (Michalewicz, 1996).

From the optimization point of view, the quality of solutions and computational complexity of the algorithms are only two parameters to be considered when comparing any two algorithms with the same starting solution. If the algorithm AL_2 produces better results than AL_1 and has a lower computational complexity (i.e., lower computational time), the algorithm AL_2 can be considered as the better algorithm irrespective of representation and operators. Of course, one has to test a sufficient number of problems including extreme cases to conclude the quality of an algorithm. In evolutionary algorithms, the same rule can be applied using equal population size as equivalent to same initial solution.

In this section, we compare two algorithms of different representations, not to be back to the debate of valid or invalid comparison, but just to demonstrate the assessment methodologies described in this chapter. The two algorithms used in this comparison are: Differential Evolution based Multiobjective Algorithms (Abbass et al., 2001), referred as *PDE* in this chapter, and Strength Pareto Evolutionary Algorithm (Zitzler and Thiele, 1999), referred as *SPEA*. *SPEA* uses binary representation with one-point crossover and bit-flip mutation and *PDE* uses real number representation with Gaussian mutation. We use only one test problem *T*1 from (Abbass et al., 2001) that was also tested by (Zitzler and Thiele, 1999).

The first method, reported in this section, is the graphical presentation of function values. The outcomes of the first five runs of the test function from each algorithm were unified. Then the dominated solutions are deleted from the union set, and all the nondominated ones are plotted. The figure is reproduced from (Abbass et al., 2001). As you can see the plot in 7.2, *PDE* clearly shows superiority over *SPEA*. From this observation, one may think that *PDE* is better than *SPEA* for any or all runs for this test problem.

The second comparison method is to measure the coverage of two solution sets produced by the two algorithms. The value of the coverage metric, calculated from 20 runs, of average 98.65 indicates that *PDE*

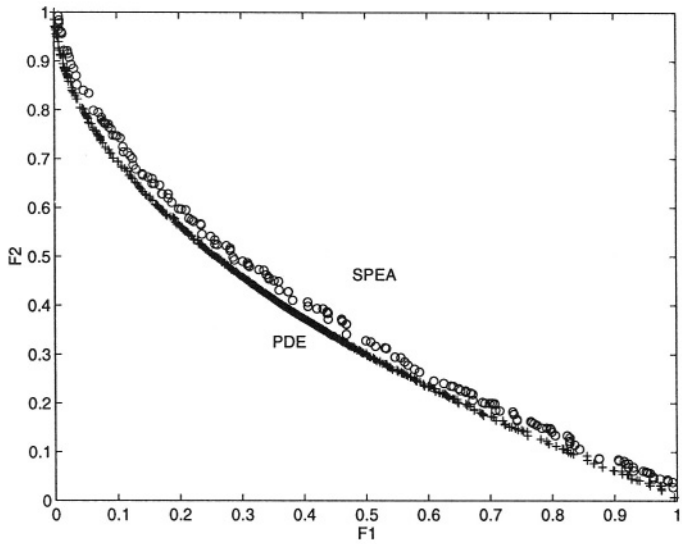


Figure 7.2. Pareto frontier for PDE and SPEA

covers *SPEA* for most cases, but not for all of them. However, *SPEA* also covers *PDE* for a few occasions. That means, *PDE* is not better than *SPEA* for all cases.

The statistical comparison, using the solutions of 20 runs, shows that $[a1,a2] = [84.30, 15.10]$. That means, *PDE* is statistically superior to *SPEA* for 84.30 percentage of the space (i.e. the percentage of lines) and *SPEA* is statistically superior to *PDE* for 15.10 percentage. The quantity $[100 - (a1 + a2)] = 0.60$, of course, gives the percentage of the space on which the results were statistically inconclusive. We use statistical significance at the 95 percent confidence level and the number of lines equal to 100. For *SPEA,* the value of statistical comparison metric is better than the coverage metric. Once again, *SPEA* is not that bad compared to *PDE* as we thought after visual inspection.

The average space coverage by *PDE* and *SPEA* are 0.341045 and 0.369344 respectively. The average spacing between the adjacent solution points are 0.1975470 and 0.2447740 for *PDE* and *SPEA* respectively. The average distance from the origin to the solution points are 0.6227760 and 0.7378490 for *PDE* and *SPEA* respectively.

From the above indicators, one can conclude easily that *PDE* solutions are better than *SPEA* for the given test problem. Although the amount of computations required by the two algorithms, to solve this

test problem, is very much similar (in terms of fitness function evaluations), the *PDE* algorithm gives special attention to the uniformity of the solution points in each generation (Abbass et al., 2001).

5. Conclusions and Future Research Paths

In this chapter, we have reviewed some of the most important proposals for metrics found in the literature on evolutionary multiobjective optimization. As we have seen, most of the proposals defined in this context normally consider only one of the three fundamental aspects that we need to measure when dealing with multiobjective optimization problems: closeness to the global Pareto front, spread along the Pareto front or number of elements of the Pareto optimal set found.

As mentioned before, several issues still remain as open paths for future research. Consider for example the following:

- It may be important to consider additional issues such as efficiency (e.g., CPU time or number of fitness function evaluations) in the design of a metric. No current metrics consider efficiency as another factor to be combined with the three issues previously discussed (it tends to be measured independently or to be considered fixed), although in some recent work by Ang et al.(2001), they propose to use a metric proposed by Feng et al. (1998) (called "optimizer overhead") for that sake. However, this metric also (implicitly) assumes knowledge of the global Pareto front of the problem when dealing with multiple objective optimization problems.
- Few metrics are really used to measure progress of MEAs through generations. Studies in which such progress is analyzed are necessary to understand better the behavior of MEAs, particularly when dealing with difficult problems.
- There is a notorious lack of comparative studies in which several metrics and MEAs are used. Such studies would not only indicate strenghts and weaknesses of MEAs, but also of the metrics used to compare them.
- There are no formal studies that indicate if measuring performance in phenotypic space (the usual norm in evolutionary multiobjective optimization) is better than doing it in genotypic space (as in operations research) (Veldhuizen, 1999). Most researchers tend to use the performance assessment measures found in the literature without any form of analysis that indicates if they are suitable for the technique and/or domain to which they will be applied.

- New metrics are of course necessary, but their type is not necessarily straightforward to determine. Statistical techniques seem very promising (da Fonseca et al., 2001), but other approaches such as the use of hyperareas looks also promising (Zitzler and Thiele, 1999). The real challenge is to produce a metric that allows to combine the three issues previously mentioned (and perhaps some others) in a non-trivial form (we saw the problems when using an aggregating function for this purpose). If such a combination is not possible, then it is necessary to prove it in a formal study and to propose alternatives (i.e., to indicate what independent measures need to be combined to obtain a complete evaluation of a MEA).
- Publicly-available test functions and results to evaluate multiobjective optimization techniques are required, as some researchers have indicated (Veldhuizen, 1999; Ang et al., 2001). It is important that researchers share the results found in their studies, so that we can build a database of approximations to the Pareto fronts of an important number of test functions (particularly, of real-world problems). The EMOO repository at: `http://www.lania.mx/~ccoello/EMOO/` is attempting to collect such information, but more efforts in that direction are still necessary.

Acknowledgments

The second author acknowledges support from the mexican Consejo Nacional de Ciencia y Tecnología (CONACyT) through project number 34201-A.

References

Abbass, H., Sarker, R., and Newton, C. (2001). PDE: A Pareto Frontier Differential Evolution Approach for Multiobjective Optimisation Problems. In *Proceedings of the Congress on Evolutionary Computation 2001,* pages 971–978. Seoul, Korea.

Ang, K., Li, Y., and Tan, K. C. (2001). Multi-Objective Benchmark Functions and Benchmark Studies for Evolutionary Computation. In *Proceedings of the International Conference on Computational Intelligence for Modelling Control and Automation (CIMCA '2001),* pages 132–139, Las Vegas, Nevada.

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

Carlyle, W. M., Kim, B., Fowler, J. W., and Gel, E. S. (2001). Comparison of Multiple Objective Genetic Algorithms for Parallel Machine Scheduling Problems. In Zitzler, E., Deb, K., Thiele, L., Coello,

C. A. C., and Corne, D., editors, *First International Conference on Evolutionary Multi-Criterion Optimization,* pages 472–485. Springer-Verlag. Lecture Notes in Computer Science No. 1993.

Caruana, R. and Schaffer, J. D. (1988). Representation and Hidden Bias: Gray vs. Binary Coding for Genetic Algorithms. In *Proceedings of the Fifth International Conference on Machine Learning,* pages 132–161, San Mateo, California. Morgan Kauffman Publishers.

Coello, C. A. C. (1999). A Comprehensive Survey of Evolutionary-Based Multiobjective Optimization Techniques. *Knowledge and Information Systems. An International Journal,* 1(3):269–308.

da Fonseca, V. G., Fonseca, C. M., and Hall, A. O. (2001). Inferential Performance Assessment of Stochastic Optimisers and the Attainment Function. In Zitzler, E., Deb, K., Thiele, L., Coello, C. A. C., and Corne, D., editors, *First International Conference on Evolutionary Multi-Criterion Optimization,* pages 213–225. Springer-Verlag. Lecture Notes in Computer Science No. 1993.

Deb, K. (1989). Genetic Algorithms in Multimodal Function Optimization. Technical Report 89002, The Clearinghouse for Genetic Algorithms, University of Alabama, Tuscaloosa, Alabama.

Deb, K. and Goldberg, D. E. (1989). An Investigation of Niche and Species Formation in Genetic Function Optimization. In Schaffer, J. D., editor, *Proceedings of the Third International Conference on Genetic Algorithms,* pages 42–50, San Mateo, California. George Mason University, Morgan Kaufmann Publishers.

Esbensen, H. and Kuh, E. S. (1996). Design space exploration using the genetic algorithm. In *IEEE International Symposium on Circuits and Systems (ISCAS'96),* pages 500–503, Piscataway, NJ. IEEE.

Eshelman, L. J., Caruana, R. A., and Schaffer, J. D. (1989). Biases in the Crossover Landscape. In Schaffer, J. D., editor, *Proceedings of the Third International Conference on Genetic Algorithms,* pages 10–19, San Mateo, California. Morgan Kaufmann Publishers.

Feng, W., Brune, T., Chan, L., Chowdhury, M., Kuek, C., and Li, Y. (1998). Benchmarks for Testing Evolutionary Algorithms. In *Proceedings of the Third Assia-Pacific Conference on Control and Measurement,* pages 134–138, Dunhuang, China.

Fogarty, T. C. (1989). Varying the Probability of Mutation in the Genetic Algorithm. In Schaffer, J. D., editor, *Proceedings of the Third International Conference on Genetic Algorithms,* pages 104–109, San Mateo, California. Morgan Kaufmann Publishers.

Fonseca, C. M. and Fleming, P. J. (1995). An Overview of Evolutionary Algorithms in Multiobjective Optimization. *Evolutionary Computation,* 3(1):1–16.

Fonseca, C. M. and Fleming, P. J. (1996). On the Performance Assessment and Comparison of Stochastic Multiobjective Optimizers. In Voigt, H.-M., Ebeling, W., Rechenberg, I., and Schwefel, H.-P., editors, *Parallel Problem Solving from Nature—PPSN IV,* Lecture Notes in Computer Science, pages 584–593, Berlin, Germany. Springer-Verlag.

Horn, J. and Nafpliotis, N. (1993). Multiobjective Optimization using the Niched Pareto Genetic Algorithm. Technical Report IlliGAl Report 93005, University of Illinois at Urbana-Champaign, Urbana, Illinois, USA.

Jaszkiewicz, A., Hapke, M., and Kominek, P. (2001). Performance of Multiple Objective Evolutionary Algorithms on a Distribution System Design Problem—Computational Experiment. In Zitzler, E., Deb, K., Thiele, L., Coello, C. A. C., and Corne, D., editors, *First International Conference on Evolutionary Multi-Criterion Optimization,* pages 241–255. Springer-Verlag. Lecture Notes in Computer Science No. 1993.

Knowles, J. D. and Corne, D. W. (2000). Approximating the Nondominated Front Using the Pareto Archived Evolution Strategy. *Evolutionary Computation,* 8(2):149–172.

Laumanns, M., Rudolph, G., and Schwefel, H.-P. (1999). Approximating the Pareto Set: Concepts, Diversity Issues, and Performance Assessment. Technical Report CI-72/99, Dortmund: Department of Computer Science/LS11, University of Dortmund, Germany. ISSN 1433-3325.

Matheron, G. (1975). *Random Sets and Integral Geometry.* John Wiley & Sons, New York.

Michalewicz, Z. (1996). *Genetic Algorithms + Data Structures = Evolution Programs.* Springer-Verlag, New York, third edition.

Ronald, S. (1997). Robust encodings in genetic algorithms. In Michalewicz, D. D. . Z., editor, *Evolutionaty Algorithms in Engineering Applications,* pages 30–44. Springer-Verlag.

Rudolph, G. (1998). On a Multi-Objective Evolutionary Algorithm and Its Convergence to the Pareto Set. In *Proceedings of the 5th IEEE Conference on Evolutionary Computation,* pages 511–516, Piscataway, New Jersey. IEEE Press.

Schott, J. R. (1995). Fault Tolerant Design Using Single and Multicriteria Genetic Algorithm Optimization. Master's thesis, Department of Aeronautics and Astronautics, Massachusetts Institute of Technology, Cambridge, Massachusetts.

Spears, W. M. and Jong, K. A. D. (1991). An Analysis of Multi-Point Crossover. In Rawlins, G. E., editor, *Foundations of Genetic Algorithms,* pages 301–315. Morgan Kaufmann Publishers, San Mateo, California.

Srinivas, N. and Deb, K. (1994). Multiobjective Optimization Using Nondominated Sorting in Genetic Algorithms. *Evolutionary Computation,* 2(3):221–248.

Veldhuizen, D. A. V. (1999). *Multiobjective Evolutionary Algorithms: Classifications, Analyses, and New Innovations.* PhD thesis, Department of Electrical and Computer Engineering. Graduate School of Engineering. Air Force Institute of Technology, Wright-Patterson AFB, Ohio.

Veldhuizen, D. A. V. and Lamont, G. B. (1998). Multiobjective Evolutionary Algorithm Research: A History and Analysis. Technical Report TR-98-03, Department of Electrical and Computer Engineering, Graduate School of Engineering, Air Force Institute of Technology, Wright-Patterson AFB, Ohio.

Veldhuizen, D. A. V. and Lamont, G. B. (1999). Multiobjective Evolutionary Algorithm Test Suites. In Carroll, J., Haddad, H., Oppenheim, D., Bryant, B., and Lamont, G. B., editors, *Proceedings of the 1999 ACM Symposium on Applied Computing,* pages 351–357, San Antonio, Texas. ACM.

Whitley, D., Rana, S., and Heckendorn, R. (1998). Representation Issues in Neighborhood Search and Evolutionary Algorithms. In Quagliarella, D., Périaux, J., Poloni, C., and Winter, G., editors, *Genetic Algorithms and Evolution Strategies in Engineering and Computer Science. Recent Advances and Industrial Applications,* chapter 3, pages 39–57. John Wiley and Sons, West Sussex, England.

Zitzler, E. (1999). *Evolutionary Algorithms for Multiobjective Optimization: Methods and Applications.* PhD thesis, Swiss Federal Institute of Technology (ETH), Zurich, Switzerland.

Zitzler, E., Deb, K., and Thiele, L. (2000). Comparison of Multiobjective Evolutionary Algorithms: Empirical Results. *Evolutionary Computation,* 8(2):173–195.

Zitzler, E. and Thiele, L. (1999). Multiobjective Evolutionary Algorithms: A Comparative Case Study and the Strength Pareto Approach. *IEEE Transactions on Evolutionary Computation,* 3(4):257–271.

This page intentionally left blank

IV

HYBRID ALGORITHMS

This page intentionally left blank

Chapter 8

UTILIZING HYBRID GENETIC ALGORITHMS

Jeffrey A. Joines and
Michael G. Kay

Abstract Genetic algorithms (GAs) have been shown to be quite effective at solving a wide range of difficult problems. They are very efficient at exploring the entire search space; however, they are relatively poor at finding the precise local optimal solution in the region in which the algorithm converges. Hybrid GAs are the combination of local improvement procedures, which are good at finding local optima, and genetic algorithms. Hybrid GAs have been shown to be quite effective at solving a wide range of problems. How the GA (the global explorer) and the local improvement procedure (the local exploiter) are combined is extremely important with respect to the final solution quality as well as the computational efficiency of the algorithm. Several different combination strategies will be investigated to determine the most effective method. Furthermore, a new adaptive memory technique will be used to enhance these methods.

Keywords: Hybrid GAs, Lamarckian Evolution, Baldwin Effect, Random Linkage, Global Optimization, Clustering Methods

1. IntrodVuction

There exists a great deal of literature on various optimization techniques. In general, it is extremely difficult to locate the optimal set of values for a system. However, for systems with special structures, e.g., convex of linear functions, etc., optimization may be relatively easy. For example, consider the optimization of the continuous, unbounded univariate function, $y = -x^2$. Simple calculus methods provide a simple solution finding the maximum of such simple functions (i.e., set the first derivative equal to zero and solve for x and then use the second deriva-

tive at that point to determine if it is a maximum, minimum, or an inflection point). For our simple function, $\frac{dy}{dx} = -2x = 0$ when $x = 0$ and the second derivative is always negative thus indicating $x = 0$ is a maximum. However, not all equations yield to such simple solutions to solve equations, e.g. $y = x\cos(x)$, $\frac{dy}{dx} = \cos(x) - x\sin(x) = 0$, which of course has an infinite number of solutions (and can not be solved).

The problem of unsolvable equations led to the development of numerical methods for searching for solutions to equations, i.e., searching for the value(s) of x that yields $\frac{dy}{dx} = 0$. A great number of these search methods have been developed, e.g., gradient search, conjugate gradient search, Newton-Raphson search, etc. However, all of these search routines only find a single root to the equation. When the equation has multiple roots, as $y = x\cos(x)$ does, the search will only find one local extreme point, not necessary the globally optimal point. For example, $y = x + 10\sin(5x) + 7\cos(4x)$ is a simple multi-modal uni-variate problem seen in Figure 8.1. If a hill-climbing technique is started at $x = 2$ or $x = 7$, the technique will climb to the top of only the current hill which represents two local maxima as seen in Figure 8.1. Obvious other starting points could be used to find the global optimal for such a simple function. This multi-start procedure will discussed later.

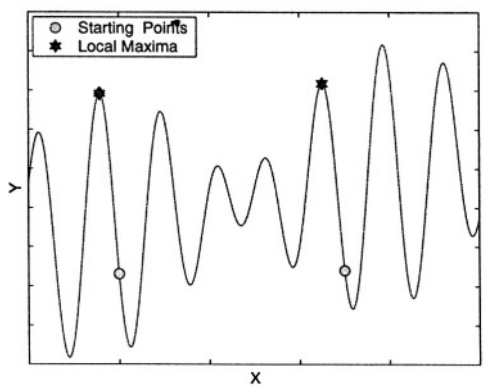


Figure 8.1. Simple Uni-variate Function

If the variables of interest are discrete (i.e., variables can only take on whole numbers), the searching process becomes even harder. Also, if the problem does not have gradients, these previous methods will not work. Exact algorithms are guaranteed to find the optimal solution (e.g., simplex algorithm for linear continuous programming problems while branch and bound/cut algorithms can be used for linear integer

program). Exact algorithms are limited to problems of certain structure or size to work. Therefore, stochastic search techniques like genetic algorithms and simulated annealing were developed as means of overcoming the local nature of more traditional search techniques as well as their limitations of certain structures, e.g., continuous and linear. Furthermore, whereas traditional search techniques use characteristics of the problem to determine the next sampling point (e.g., gradients, Hessians, linearity, and continuity), stochastic search techniques make no such assumptions. Instead, the next sampled point(s) is(are) determined based on stochastic sampling/decision rules rather than a set of deterministic decision rules.

A large number of heuristic search algorithms are available for function optimization, which provide (possibly) suboptimal solutions in polynomial time with no guarantee on their quality. For example, local improvement procedures (LIPs) search a "neighborhood" of starting solution until either first improvement or best improvement (local optimum) is found. Recently, a lot of attention has been directed at exploring the efficiency of meta-heuristics for solving hard search problems. Meta-heuristics guide the application and use of local heuristics, e.g., one heuristic might determine the starting point for a lower level heuristic. They are used to search the many local optima which most local heuristics find, attempting to locate the global optimum. This approach has met with a good deal of success (Golver and Laguna, 1997; Joines and Culbreth, 1999; Michalewicz, 1996). Each of these heuristics, e.g., simulated annealing, genetic algorithms, tabu search, etc., has been shown to be effective at finding good solutions efficiently. Random restart (multistart), the simplest meta-heuristic, generates a set of random starting solutions, applies the LIP to each of these starting solution, and reports the best solution found. However, no past information is utilized to guide the search. Tabu search a meta-heuristic often used for combinatorial problems retains memory of solutions previously visited via or a tabu list of current LIP neighborhood. The list is used to avoid cycling and memory outside of neighborhood used to promote diversification and intensification strategies. The concepts of long term memory in GAs will be addressed in Section 3.1. Tabu search has been shown to out perform GAs on certain classes of problems (Golver and Laguna, 1997; Joines et al., 2000b), likewise, GAs have been shown to out perform simulated annealing (Houck et al., 1996b) as well as Tabu search (Joines et al., 2000b). Therefore, according to the "No Free Lunch" theorem (Wolpert and Macready, 1997), algorithms which are tuned for

one problem do well for these problems but will perform poorly on other classes of problems.

This chapter presents an empirical analysis of the use of genetic algorithms utilizing local improvement procedures to solve a variety of test problems. Several different hybridization schemes will be explored. Section 2 describes how GAs can be hybridized by incorporating local improvement procedures into the algorithm, Incorporating local improvement procedures gives rise to the concepts of Larmarckian evolution, the Baldwin Effect, and partial Lamarckianism, all of which are explained in Section 2.1. Also, the concept of a one-to-one genotype to phenotype mapping is reviewed, where genotype refers to the space the GA searches while the phenotype refers to the space of the actual problem. The first set of experiments applies these concepts to three different classes of problems described in Section 2.2: five different multi-modal nonlinear function optimization problems, the location-allocation problem, and the manufacturing cell formation problem. Section 2.3 presents the results of a series of empirical tests of the various search strategies described in Section 2.1 and a partial Lamarkian method is shown to be the most effective in terms of solution quality and computational efficiency. However, even the Lamarkian schemes waste valuable computational time by repeating local searchers in the same basins of attraction (i.e., local minima). Therefore, Section 3 presents a hybridization method that utilizes adaptive memory to overcome this limitation. In Section 3.1, Random Linkage (a stochastic search technique that utilizes adaptive memory to determine when to apply the local search) is explained. Section 3.2 describes EARL, which is the combination of the adaptive memory procedure in Random Linkage with a hybrid genetic algorithm to produce a more computational efficiency algorithm. The second set of experiments in Section 3.3 demonstrates that EARL is more computational efficient in finding the best solution as compared to the best Lamarkian schemes and pure Random Linkage.

2. Hybridizing GAs with Local Improvement Procedures

Many researchers (Chu and Beasley, 1995; Davis, 1991; Houck et al., 1997; Houck et al., 1996a; Joines et al., 2000a; Michalewicz, 1996; Renders and Flasse, 1996) have shown that GAs perform well for global searching because they are capable of quickly finding and exploiting promising regions of the search space, but they take a relatively long time to converge to a local optimum. For example, Figure 8.2 shows three replications of a typical GA run for the 100×40 manufacturing cell

formation problem described in Section 2.2 (i.e., 140 nonlinear integer variable problem) of Chandrasekharan and Rajagopalan (1987) (referred to as Chan). The GA quickly finds a good solution but requires many more generations to reach the optimal solution (or the best known solution). Table 8.1 shows the number of generations for each replication along with the mean number of operations to reach a certain percent of optimal. On average, the GA reaches 90% of optimal after only 65% of the total computational time. It takes approximately a third of the time to go from 90% of optimal to exact optimal. Michalewicz (1996) and Joines (1996c) developed two mutation operators to help increase the exploitation of the local solution. However, these methods still fall short of helping with the local exploitation.

Table 8.1. The Number of Generations Needed to Reach % Optimal for Chan Dataset

% of Optimal	Repl. 1 Gen(%Gen)	Repl. 2 Gen(%Gen)	Repl. 3 Gen(%Gen)	Mean Gen(%Gen)
10	2 (0.2)	5 (0.3)	3 (0.2)	3.3 (0.2)
25	108 (10.4)	95 (6.1)	89 (5.2)	97.3 (6.8)
50	294 (28.3)	265 (16.9)	256 (15.0)	271.7 (18.9)
75	440 (42.3)	441 (28.2)	884 (51.7)	588.3 (40.9)
90	623 (60.0)	1021 (65.2)	1173 (68.6)	939.0 (65.3)
95	657 (63.2)	1233 (78.7)	1275 (74.5)	1055.0 (73.3)
99	855 (82.3)	1566 (100.0)	1711 (100.0)	1377.3 (95.7)
100	1039 (100.0)	1566 (100.0)	1711 (100.0)	1438.7 (100.0)

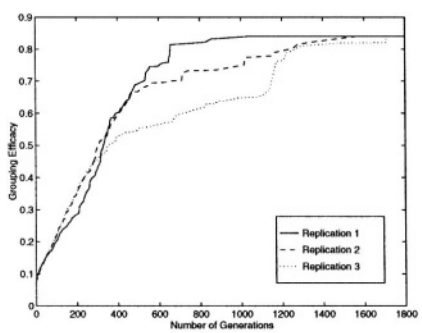


Figure 8.2. The Solution Quality of the GA versus Generations for Chan Dataset

Local improvement procedures (LIPs), e.g., two-opt switching for combinatorial problems and gradient descent for unconstrained nonlinear problems, quickly find the local optimum of a small region of the search space, but are typically poor global searchers. Because these

procedures do not guarantee optimality, in practice, several random starting points are generated and used as input into the local search technique and the best solution is recorded. This global optimization technique (multistart) has been used extensively but is a blind search technique since it does not take account past information (Houck et al., 1996a; Rinnooy-Kan and Timmer, 1987; Locatelli and Schoen, 1999). GAs, unlike multistart, utilize past information in the search process. Therefore, LIPs have been incorporated into GAs in order to improve their performance through what could be termed "learning." Such hybrid GAs have been used successfully to solve a wide variety of problems (Chu and Beasley, 1995; Houck et al., 1997; Houck et al., 1996a; Joines et al., 2000a; Joines et al., 2000b; Michalewicz, 1996; Renders and Flasse, 1996). Houck et al. (1995) showed that for the continuous location–allocation problem, a GA that incorporated a LIP outperformed multistart and a two-way switching procedure, where both methods utilized the same LIP as the hybrid GA.

There are several ways LIPs can be incorporated into a GA. The particular combination is extremely important in terms of possible solution quality and computational efficiency. We need to find the right mix of local exploitation versus global exploration. The following methods are used in the experimentation described in Section 2.2 in order to gain insight into which implementation method is best.

1 Run the GA without the LIP and then apply it to the final solution obtained by the GA. This allows the precise local optimum around the final solution to be found.

2 Use the LIP as the GA's evaluation function. (Recall, the GA makes no assumptions on the form of the objective, only that it map the individuals in the population into a totally ordered set.) The LIP is used to evaluate individuals in the population (i.e., determine the best objective value for this starting assignment). In this context, the GA generates starting location for the LIP similar to the multi-start procedure. However, past information is being used to drive the global searching.

3 Apply the LIP as a genetic operator in the same manner as any mutation operator. In this approach, the LIP is only applied to a small portion of the parents (i.e., the parents selected to undergo this type of mutation) rather than all of the children created. Chu and Beasily (1995) used a LIP as a genetic operator for the generalized QAP. For the generalized QAP, the LIP was computational expensive and applying it only a few times per generation was

best. However, Joines and other (Joines et al., 1996d; Joines et al., 2000a) showed for several versions of the manufacturing cell formation problem (an non-linear integer programming problem), that LIP genetic operator was not as good as utilizing it directly as an evaluation function.

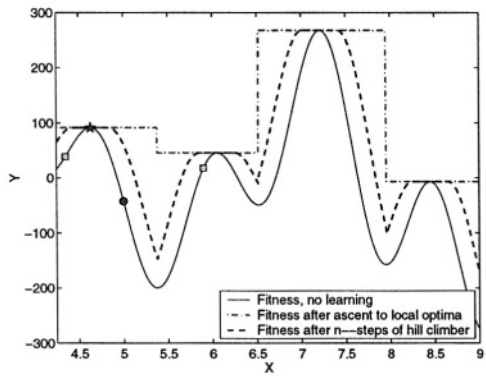
Incorporating a LIP as an evaluation function gives rise to the concepts of the Baldwin Effect and Lamarckian evolution. Also, the concept of a one-to-one genotype to phenotype mapping is introduced, where genotype refers to the space the GA searches while the phenotype refers to the actual problem space.

2.1 The Baldwin Effect and Lamarckian Evolution

Local improvement procedures have been incorporated into GAs in order to improve the algorithm's performance through learning. There are two basic models of evolution that have been used to incorporate learning into a GA: the Baldwin Effect and Lamarckian evolution. The Baldwin Effect allows an individual's fitness (phenotype) to be determined based on learning, i.e., the application of local improvement. Like natural evolution, the result of the improvement does not change the genetic structure (genotype) of the individual, it just increases the individual's chances of survival. Lamarckian evolution, in addition to using learning to determine an individual's fitness, changes the genetic structure of an individual to reflect the result of the learning. Both "Baldwinian" and "Lamarckian" learning have been investigated in conjunction with hybrid GAs.

2.1.1 Baldwin Effect. The Baldwin Effect, as utilized in GAs, was first investigated by Hinton and Nolan (1987) using a flat landscape with a single well representing the optimal solution. Individuals were allowed to improve by random search, which in effect transformed the landscape to include a funnel around the well. They showed that, without learning, the GA fails; however, with the random search, the GA is capable of finding the optimum.

Whitley et al. (1994) demonstrated that "exploiting the Baldwin Effect need not require a needle in a haystack and improvements need not be probabilistic." They showed that a LIP can, in effect, change the landscape of the fitness function into flat landscapes around the local basins (see Figure 8.3). This transformation increases the likelihood of allocating more individuals to certain basins, i.e., we are conceptually searching a step function. It is much easier for an algorithm to hit a step



Type of Learning	Before Learning x	Before Learning f(x)	After Learning x	After Learning f(x)
Baldwinian Learning	5.00	-42.49	5.00	91.56
Lamarckian Learning	5.00	-42.49	4.63	91.56

Figure 8.3. Transformed Fitness Landscape using Learning (Adapted from Whitley et al., 1994)

then the precise local minimum. Reaching the local optima from every point in the search space may be computational unattractive. However, as seen in Figure 8.3, allowing the LIP to perform only n steps produces smaller fiat regions around the basins of attraction, giving similar benefits. This allows the user to control the amount of global exploration versus local exploitation. In a comparison of Baldwinian and Lamarckian learning, Whitley et al. (1994) showed that utilizing either form of learning is more effective than the standard GA approach without the LIP (a bitwise steepest ascent algorithm performed on a binary representation). They argued that, while Lamarckian learning is faster, it may be suspectable to premature convergence to a local optimum as compared to Baldwinian learning. Their results were inconclusive.

2.1.2 Lamarckian Evolution. Lamarckian learning forces the genotype to reflect the result of some form of local improvement (see Figure 8.3). This results in the inheritance of acquired or learned characteristics that are welladapted to the environment. The improved individual is placed back into the population and allowed to compete for reproductive opportunities. However, Lamarckian learning inhibits the schema processing capabilities of genetic algorithms (Whitley et al., 1994). Changing the genetic information in the chromosomes results

in a loss of inherited schema, altering the statistical information about hyperplane partitions implicitly contained in the population.

While Lamarckian learning may disrupt the schema processing of a genetic algorithm, Baldwinian learning certainly aggravates the problem of multiple genotype to phenotype mappings. A GA works on both genotypes and phenotypes. A genotype refers to the composition of the values in the chromosome or individual in the population, whereas a phenotype refers to the solution that is constructed from a chromosome. In a direct mapping, there is no distinction between genotypes and phenotypes. For example, to optimize the function $5\cos(x_1) - \sin(2x_2)$, a typical representation for the chromosome would be a vector of real numbers (x_1, x_2), which provides a direct mapping to the phenotype. However, for some problems, a direct mapping is not possible or desired (Nakano, 1991). The most common example of this is the traveling salesperson problem with an ordinal representation. Here, the genotype is represented by an ordered list of cities to visit. The phenotype is a tour, and any rotation of the chromosome yields the same tour; thus, any rotation of a genotype results in the same phenotype. For example, the two tours (1,2,3,4) and (3,4,1,2) have different genotypes since their gene strings are different, but both strings represent the same tour and thus have the same phenotype.

It has been noted that having multiple genotypes map to the same phenotype may confound the GA (Hinton and Nolan, 1987; Nakano, 1991). This problem also occurs when an LIP is used in conjunction with a GA. Consider the example of maximizing $\sin(x)$. Suppose a simple gradient-based LIP is used to determine the fitness of a chromosome. Then any genotype between $[-\pi/2, 3\pi/2]$ will have the same phenotype value of 1.

2.1.3 Partial Lamarckianism. Hybrid genetic algorithms need not be restricted to operating in either a pure Baldwinian or pure Lamarckian manner. Instead, a mix of both strategies, or what is termed "partial Lamarckianism" (Houck et al., 1997; Joines et al., 2000a) could be employed. For example, a possible strategy is to update the genotype to reflect the resulting phenotype in 50% of the individuals. While this 50% partial Lamarckian strategy has no justification in natural evolution, for simulated evolution this mix is as valid as either pure Lamarckian or pure Baldwinian search.

Orvosh and Davis (1994) advocated the use of a 5% rule for updating individuals when employing repair functions in genetic algorithms to solve constrained optimization problems. All infeasible solutions generated by the genetic algorithm are repaired to the feasible domain in

order to determine their fitness. The 5% rule dictates that 5% of the infeasible individuals have their genetic representation updated to reflect the repaired feasible solution. This partial updating was shown on a set of combinatorial problems to be better than either no updating or always updating. However, Michalewicz (1996) determined that a higher percentage update 20–25% did better when using repair functions to solve continuous constrained nonlinear programming problems.

Previous research has concentrated either on the comparison of pure Lamarckian and pure Baldwinian search, or the effectiveness of partial repair for constrained optimization. This chapter examines the use of partial Lamarckianism with regard to the use of LIPs on the cell formation problem(Houck et al., 1997).

2.2 First Set of Experiments

To investigate the trade-off of disrupted schema processing in Lamarckian learning and of multiple genotype mapping to the same phenotype in Baldwinian learning, a series of experiments using LIPs as evaluation functions and as genetic operators were performed on several different cell formation test problem instances. For each test problem instance, the GA was run with varying levels of Lamarckianism from 0% (pure Baldwinian) to 100% (pure Lamarckian) to determine if there is a trend between the two extremes, or if combinations of Baldwinian learning and Lamarckian learning were beneficial. In these experiments, individuals are updated to match the resulting phenotype with a probability of 0, 5, 10, 20, 30, 40, 50, 60, 80, 90, 95, 100%. The GA was also run using the LIP as a mutation operator instead to determine if this method of hybridization is better in terms of computational efficiency and/or quality of solution. In these experiments, the number of LIP mutations performed was 3, 4, 5, or 6. A pure genetic approach, i.e., no local improvement, was used for comparison purposes.

Each run of the GA was replicated 30 times, with common random seeds. The genetic algorithm parameters used in these experiments are defined in Table 8.2. The genetic algorithm is of a form advocated by Michalewicz (1996) and uses a floating point (real-valued) representation with three real-valued crossovers (simple, heuristic, and arithmetic) and five real-valued mutations (boundary, uniform, multi-uniform, non-uniform, and multi-non-uniform). However, for the cell formation problems, the GA is modified to work with integer variables (Joines et al., 1994b). A normalized geometric ranking scheme used as the selection procedure (Houck et al., 1997). The GA was terminated when either the optimal or best known solution was found or after one million function

evaluations were performed. Using function evaluations allows a direct comparison between the hybrid GA methods and the pure GA (i.e., no local improvement) because it takes into account the cost of using a LIP (i.e., the number of function evaluations generated by the LIP)

Table 8.2. Parameters Used in the Experiments

Parameter	Value
Boundary Mutation Operators	4
Uniform Mutation Operators	4
Multi-Uniform Mutation Operators	4
Non-Uniform Mutation Operators	4
Multi-Non-Uniform Mutation Operators	8
Arithmetic Crossover Operators	6
q, the prob. of selecting the best individual	0.08
Maximum no. of Function Evaluations	1,000,000
Population Size	80

Multiple instances of seven different test problems were used in this investigation to ensure that any effect observed (1) held across different classes of problems as well as different sized instances of the problem and (2) was not due to some unknown structure of the specific class of problem selected, specific instance chosen, or the particular local improvement procedure used. The first five test problems are bounded nonlinear optimization test problems taken from the literature, while the last two problems are the location-allocation and manufacturing cell formation problems. For each of the nonlinear optimization problems, three different size instances (n = 2, 10, 20) were used in the study. For both the location-allocation and cell formation problems, two different sized instances were used.

2.2.1 Brown's Almost Linear Function. The first test problem comes from a standard test set of nonlinear optimization problems (More et al., 1981) and is as follows:

$$f(x) = \sum_{i=1}^{n}((f_i(x))^2, \qquad for x_i \in [-25, 25] \qquad (8.1)$$

where

$$f_i(x) = \begin{cases} x_i + \sum_{j=1}^{n} x_j - (n+1), & for i = 1, ..., n-1 \\ \prod_{j=1}^{n} x_j - 1, & \text{for } i = n \end{cases}$$

The function is not linearly separable and has the basic form of a nonlinear least squares problem. Three different instances of this problem were used for this investigation: a 2-dimensional instance (Brown-2), a 10-dimensional instance (Brown-10), and a 20-dimensional instance (Brown-20).

2.2.2 Corana Function. The next test problem consists of three instances of the modified Corana problem: a 2-, 10-, and 20-dimensional instance (Corana-2, Corana-10, and Corana-20, respectively). This problem comes from a family of continuous nonlinear multimodal functions developed by Corana et al. (1987) to test the efficiency and effectiveness of simulated annealing as compared to stochastic hill-climbing and Nelder-Mead optimization methods. This parametrized family of test functions contains a large number of local minima. The function can be described as an n-dimensional parabola with rectangular pockets removed and with the global optimum always occurring at the origin. These functions are parameterized with the number of dimensions (n), the number of local optima, and the width and height of the rectangular pockets. As formulated in Corana et al. (1987), the function proved to be too easy: all the learning methods found the optimal solution every time with the same amount of computational effort. Therefore, the first condition in the definition of $g(x_i)$ below was modified so that rectangular troughs, instead of pockets, are removed from the function. This was accomplished by eliminating the restriction that all, instead of any, x_i satisfy and all k_i are not zero. The modified Corana function is as follows:

$$f(x) = \sum_{i=1}^{n} c_i g(x_i), \text{for } c_i \in c \text{ and } x_i \in [-10000, 10000] \tag{8.2}$$

where

$$g(x_i) = \begin{cases} 0.15 z_i^2, & \text{if } k_i s_i - t_i \leq x_i \leq k_i s_i + t_i, \text{ for any integer } k_i \\ x_i^2, & \text{otherwise} \end{cases}$$

$$z_i = \begin{cases} k_i s_i + t_i & \text{if } k_i < 0 \\ 0 & \text{if } k_i = 0 \\ k_i s_i - t_i & \text{if } k_i > 0 \end{cases}$$

$$c = (1, 1000, 10, 100, 1, 10, 100, 1000, 1, 10, \ldots \\ 100, 1000, 1, 10, 100, 1000, 1, 10, 100, 1000)$$

$$s_i = 0.2$$

$$t_i = 0.05$$

2.2.3 Griewank Function. The Griewank function is the third test problem, again with three instances of 2, 10, and 20 dimensions, respectively (Griewank-2, Griewank-10, and Griewank-20). The Griewank function is a bounded optimization problem used by Whitley et al. (1994) to compare the effectiveness of Lamarckian and Baldwinian search strategies. The Griewank function, like Brown's almost linear function, is not linearly separable, and is as follows:

$$f(x) = \sum_{i=1}^{n} \frac{x_i^2}{4000} - \prod_{i=1}^{n} \cos(\frac{x_i}{\sqrt{i}}) - 1, \text{ for } x_i \in [-512, 511] \qquad (8.3)$$

2.2.4 Rastrigin Function. The next test problem, with three instances of 2, 10, and 20 dimensions (Rastrigin-2, Rastrigin-10, and Rastrigin-20), is also a bounded minimization problem taken from Whitely et al. (1994). The functional form is as follows:

$$f(x) = 10n + \sum_{i=1}^{n} (x_i^2 - 10\cos(2\pi x_i)), \text{ for } x_i \in [-5.12, 5.11] \qquad (8.4)$$

2.2.5 Schwefel Function. The last nonlinear test problem, with three instances of 2, 10, and 20 dimensions (Schwefel-2, Schwefel-10, and Schwefel-20), is also a bounded minimization problem taken from Whitley et al. (1994). The functional form is as follows:

$$f(x) = nV + \sum_{i=1}^{n} -x\sin(\sqrt{|x_i|}), \text{for } x_i \in [-512, 511] \qquad (8.5)$$

where

$$V = \min\{-x\sin(\sqrt{|x_i|}), \qquad x \in [-512, 511]$$

V depends on system precision; for our experiments,
$V = 418.9828872721625$.

For these five nonlinear optimization test problems, a sequential quadratic programming (SQP) optimization method was used as the LIP. SQP is a technique used to solve constrained nonlinear programming problems. SQP works by first computing the descent direction by solving a standard quadratic program using a positive definite estimate of the Hessian of the Lagrangian. A line search is then performed to find the feasible minimum of the objective along the search direction. This process is repeated until the algorithm converges to a local minimum, or exceeds a certain number of iterations. For the purposes of this study, SQP was stopped after 25 iterations if it had not found a local optimum. Therefore, we are limiting the amount of local exploitation. A full description of the SQP used in this study can be found in (Lawrence et al., 1994).

2.2.6 Location-Allocation Problem. The continuous location -allocation problem, as described in (Houck et al., 1996b), is used as the next test problem. The location-allocation (LA) problem, a type of nonlinear integer program, is a multifacility location problem in which both the location of n new facilities (NFs) and the allocation of the flow requirements of m existing facilities (EFs) to the new facilities are determined so that the total transportation costs are minimized. The location-allocation problem is a difficult optimization problem because its objective function is neither convex nor concave, resulting in multiple local minima (Cooper, 1972). Optimal solution techniques are limited to small problem instances (less than 25 EFs for general l_p distances (Love and Juel, 1982) and up to 35 EFs for rectilinear distances (Love and Juel, 1982)). Two rectilinear distance instances of this problem were used in the investigation: a 200 EF by 20 NF instance (LA-200) and a 250 EF by 25 NF instance (LA-250), both taken from (Houck et al., 1996b).

Given m EFs located at known points $a_j, j = 1, ..., m$, with associated flow requirements $w_j, j = 1, ..., m$, and the number of NFs, n, from which the flow requirements of the EFs are satisfied, the location-allocation problem can be formulated as the following nonlinear program:

$$\text{Minimize} \quad \sum_{i=1}^{n}\sum_{j=1}^{m} w_{ij} d(X_i, a_j) \tag{8.6}$$

$$\text{Subject to} \quad \sum_{i=1}^{n} w_{ij} = w_j, j = 1, \ldots, m$$

$$w_{ij} \geq 0, i = 1, \ldots, n; j = 1, \ldots, m$$

where the unknown variable locations of the NFs, $X_i, i = 1, ..., n$, and the flows from each NF i to each EF $j, w_{ij}, i = 1, ..., n, j = 1, ..., m$, are the decision variables to be determined, and $d(X_i, a_j)$ is the distance between NF i and EF j. An infinite number of locations are possible for each NF i since each X_i is a point in a continuous, typically two-dimensional, space.

The genetic algorithm approach to solve this problem, described in detail in (Houck et al., 1996b), is used in these experiments. The individuals in the GA are a vector of real values representing the starting locations for the new facilities. Based upon these locations for the new facilities, the optimal allocations are found by allocating each existing facility to the nearest new facility. The total cost of the resulting network is then the weighted sum of the distances from each existing facility to its allocated new facility. The alternate location-allocation method developed by Cooper (1972) is used as the LIP for this problem. This method quickly finds a local minimum solution given a set of starting locations for the new facilities. This procedure works by starting with a set of new facility locations; it then determines an optimal set of allocations based on those locations, which for this uncapacitated version of the problem reduces to finding the closest new facility to each existing facility. The optimal new facility locations for these new allocations are then determined by solving n single facility location problems, which is in general a nonlinear convex optimization problem. This method continues until no further allocation changes can be made.

2.2.7 Cell Formation Problem. The manufacturing cell formation problem, as described in (Joines et al., 2000a; Joines et al., 1994b; Joines et al., 1996d), is the final test problem. The cell formation problem, a type of nonlinear integer programming problem, is concerned with assigning a group of machines to a set of cells and assigning a series of parts needing processing by these machines into families. The goal is to create a set of autonomous manufacturing units that eliminate inter-cell movements (i.e., a part needing processing by a machine outside its assigned cell). Two instances of this problem were in the investigation:

a 22 machine by 148 part instance (Cell-1) and a 40 machine by 100 part instance (Cell-2).

Consider an m machine by n part cell formation problem instance with a maximum of k cells. The problem is to assign each machine and each part to one and only one cell and family, respectively. Joines et al. (1994b) developed an integer programming model with the following set variable declarations that eliminate the classical binary assignment variables and constraints:

$$x_i = l, \qquad \text{if machine } i \text{ is assigned to cell } l$$
$$y_j = l, \qquad \text{if part } j \text{ is assigned to family } l$$

The model was solved using a GA. In the GA, the individuals are a vector of integer variables representing the machine and part assignments $(x_1, x_2, ..., x_m, y_1, y_2, ..., y_n)$. This representation has been shown to perform better than other cell formation methods utilizing the same objective (Joines et al., 1996c). The following "grouping efficacy" measure (Γ) is used as the evaluation function:

$$\text{Maximize} \quad \Gamma = \frac{e - e_o}{e + e_v} \tag{8.7}$$

where

$$e = e_o + e_d$$
$$e_v = \sum_{i=1}^{k} D_i - d_i$$

Grouping efficacy tries to minimize a combination of both the number of exceptional elements, e_o (i.e., the number of part operations performed outside the cell), and the process variation, e_v (i.e., the discrepancy of the parts' processing requirements and the processing capabilities of the cell). Process variation for each cell i is calculated by subtracting from the total number of operations a cell can perform, D_i, the number part operations that are performed in the cell, d_i. The total process variation is the sum of the process variation for each cell. Exceptional elements decrease since the material handling cost for transportation within a cell is usually less than the cost of the transportation between cells. Process variation degrades the performance of the cell due to increased scheduling, fixturing, tooling, and material handling costs.

A greedy one-opt switching method, as described in (Joines et al., 2000a), was used as the LIP. Given an initial assignment of machines

and parts to cells and families, respectively, the LIP uses a set of improvement rules developed by Ng (1993) to determine the improvement in the objective function if machine i is switched from its current cell to any of the remaining $k-1$ cells. The procedure repeats this step for each of the m machines as well as each of the n parts, where each part is switched to $k-1$ different families.

2.3 Summary of Results

Because GAs have few theoretical properties, computation experimentation is performed to validate an algorithm's effectiveness. Therefore, to ensure correct conclusions, careful experimentation and statistical analysis needs to be performed. For more information on constructing and analayzing experiments, see the studies by Barr et al. (1995), Hooker (1995), and Houck et al. (1997).

For the 30 replications of each problem instance, both the mean number of function evaluations and the mean final function values were determined. To ensure the validity of the subsequent analysis of variance (ANOVA) and multiple means tests, each data set was tested first for normality and second for homogeneity of the variances among the different search strategies. The Shapiro-Wilk test for normality was performed on the results at $\alpha = 0.05$. Since there is no established test for variance homogeneity (SAS, 1990), visual inspection was used to determine if there were significant variance differences. If there were significant differences and there existed a strong linear relationship between the means and standard deviations (i.e., a linear regression correlation greater than 0.8), a logarithmic transformation was used. For the number of functional evaluations, standard deviations equal to zero (which correspond to the search strategy terminating after one million functional evaluations), were removed from the linear regression since these values were not due to randomness. For those results passing the Shapiro-Wilk test for normality, an ANOVA was conducted for both the number of function evaluations and the final functional value. If the ANOVA showed a significant effect, $\alpha > 0.01$, the means of each of the 12 different search strategies considered, namely, no local improvement (referred to as N), pure Baldwinian (referred to 0), pure Larmarckian (100), and nine levels of partial Lamarckian (referred to as 5, 10, 20, 40, 50, 60, 80, 90, and 95, respectively), were compared using a multiple means comparison methods. Three different statistical methods exist for performing a multiple means comparison: the Duncan approach to minimize the Bayesian loss function; Student-Newman-Keuls (SNK), a multiple range test using a multiple stage approach; and an approach developed by Ryan, Einot and

Gabriel, and Welsch (REGW), which is also a multiple stage approach that controls the maximum experimentwise error rate under any complete or partial hypothesis. Each of the tests were conducted using the general linear models routine in SAS v6.09 (SAS, 1990). There is no consensus test for the comparison of multiple means. In general, all the tests could be ran to determine the groups (Houck et al., 1997; Joines et al., 2000a). For this chapter, the SNK test was chosen.

The multiple means composition methods are statistical tests that provide information on sets of means whose differences are statistically significant. The strategies assigned to the best group are shaded. The rank indicates the rank of the various means where one is better. To concisely present the main results of the SNK statistical test, Tables 8.3 and 8.4 show rankings of each search strategy with respect to the fitness of the final result and the number of function evaluations required, respectively. For each strategy, the rankings were determined by finding the group which yielded the best results for each test problem instance as determined by the SNK means analysis. If the strategy was placed into a single group, that group's rank was used; however, if the method was placed into several overlapping groups, the method was placed into the group with the best rank. Therefore, these rankings represent how many groups of means are significantly better for each test problem instance. The SNK multiple means test was arbitrarily chosen for these rankings. Table 8.3 shows the rankings for the final fitness of the solutions returned by each of the search strategies, where 1 represents the best rank and is shaded, 2 represents the next best rank, etc. All of the strategies employing at least 20% Lamarckian learning consistently found the optimal solution. However, the pure Baldwinian (0%) and the 5 and 10% partial Lamarckian strategies did find significantly worse solutions for several test problem instances. The GA employing no local improvement procedure (N) is included to provide a comparison with how efficiently the local search procedure utilizes function evaluations as compared to a pure genetic sampling approach. For most of these test problem instances, the use of the local improvement procedure (LIP) significantly increases the efficiency of the genetic algorithm.

Table 8.4 shows the rankings for the number of function evaluations required to locate the final functional value returned by each of the search strategies. The table shows an interesting trend: neither the pure Lamarckian (100) nor pure Baldwinian (0) strategies consistently yielded the best performance; this was also observed by Whitley et al. (1994). Using a binary representation and a bitwise steepest ascent LIP, Whitley et al. (1994) demonstrated that a Baldwinian search strategy was preferred for the 20-dimensional Schwefel function while the results

Table 8.3. Rank of Solutions (SNK)

Problem Instance	Search Strategy											
	N	0	5	10	20	40	50	60	80	90	95	100
Brown-2†	1	1	1	1	1	1	1	1	1	1	1	1
Brown-10	3	2	1	1	1	1	1	1	1	1	1	1
Brown-20	3	2	1	1	1	1	1	1	1	1	1	1
Corana-2†	1	1	1	1	1	1	1	1	1	1	1	1
Corana-10	2	4	3	2	1	1	1	1	1	1	1	1
Corana-20	4	2	8	7	6	5	4	3	2	1	1	1
Griewank-2†	2	1	1	1	1	1	1	1	1	1	1	1
Griewank-10	2	1	1	1	1	1	1	1	1	1	1	1
Griewank-20	2	1	1	1	1	1	1	1	1	1	1	1
Rastrigin-2†	1	1	1	1	1	1	1	1	1	1	1	1
Rastrigin-10†	2	1	1	1	1	1	1	1	1	1	1	1
Rastrigin-20	2	1	1	1	1	1	1	1	1	1	1	1
Schwefel-2†	1	1	1	1	1	1	1	1	1	1	1	1
Schwefel-10	1	2	1	1	1	1	1	1	1	1	1	1
Schwefel-20	1	2	1	1	1	1	1	1	1	1	1	1
LA-200	3	2	1	1	1	1	1	1	1	1	1	1
LA-250	3	2	1	1	1	1	1	1	1	1	1	1
Cell-1	3	2	1	1	1	1	1	1	1	1	1	1
Cell-2	2	2	1	1	1	1	1	1	1	1	1	1
†- Failed Shapiro-Wilk test, or ANOVA did not show significant effect at $\alpha = 0.01$												

in Table 8.4 show a preference for a Lamarckian search strategy when using a floating point representation and SQP. Also, a Lamarckian strategy was preferred in Whitley et al. (1994) for the 20-dimensional Griewank instance, whereas in this study a Baldwinian search strategy is preferred.

The results of this study and those of Whitley et al. (1994), together, show that a combination of the problem, the representation, and the local improvement procedure (LIP) all can influence the effectiveness of either a pure Lamarckian or pure Baldwinian search strategy. However, with respect to the representations and LIPs used in this study, a partial Lamarckian strategy tends to perform well across all of the problems considered. Taking a minimax approach, i.e., minimizing the worst possible outcome, the 20% and 40% partial Lamarckian search strategies provide the best results. The worst ranking of each search strategy, in terms of convergence speed, is shown as the last row of Table 8.4. Both the 20% and 40% strategies yield, at worst, the third best convergence to the optimal solution. The other search strategies, 0% and 5% partial Lamarckian did not consistently find the optimal. Higher amounts

Table 8.4. Rank of Convergence Speed (SNK)

Problem Instance	Search Strategy											
	N	0	5	10	20	40	50	60	80	90	95	100
Brown-2	2	1	1	1	1	1	1	1	1	1	1	1
Brown-10	6	5	4	3	2	1	1	1	1	1	1	1
Brown-20	4	4	3	2	1	1	1	1	1	1	1	1
Corana-2	2	1	1	1	1	1	1	1	1	1	1	1
Corana-10	4	4	4	4	3	2	2	2	2	1	1	1
Corana-20	3	3	3	3	3	3	3	2	2	1	1	2
Griewank-2†	1	1	1	1	1	1	1	1	1	1	1	1
Griewank-10	7	1	1	1	2	3	4	4	5	5	5	6
Griewank-20	5	1	1	1	1	2	3	3	3	4	4	4
Rastrigin-2	2	1	1	1	1	1	1	1	1	1	1	1
Rastrigin-10	3	5	4	4	3	2	1	1	1	1	1	1
Rastrigin-20	2	6	5	4	3	1	1	1	1	1	1	1
Schwefel-2†	1	1	1	1	1	1	1	1	1	1	1	1
Schwefel-10	1	3	2	2	1	1	1	1	1	1	1	1
Schwefel-20	3	4	2	2	1	1	1	1	1	1	1	1
LA-200	3	2	1	1	1	1	1	1	1	1	1	1
LA-250	3	2	1	1	1	1	1	1	1	1	1	1
Cell-1	2	2	1	1	1	1	1	1	1	1	1	1
Cell-2	2	2	1	1	1	1	1	1	1	1	1	1
Worst Rank	7	6	5	4	3	3	4	4	5	5	5	6

†- Failed Shapiro-Wilk test,
or ANOVA did not show significant effect at $\alpha = 0.01$

of Lamarckianism, 50% to 100%, converge slower to the optimal solution. Since no single search strategy is dominant in terms of solution quality and computational efficiency, the worst case performance of the search strategy can be minimized across the entire set of test problems examined by employing a 20% to 40% partial Lamarckian strategy. The no local improvement search strategy (N) is shown for comparison to demonstrate the effectiveness of hybrid genetic algorithms.

3. Adaptive Memory GA's

In the previous section, it was shown that Baldwinian learning aggravates the problem of multiple genotype to phenotype mappings. If two individuals are different but map to the same local basin, the GA will try to exponentially exploit both individuals especially since their fitness would be the same through learning. If these two individuals are crossed over to produce offspring in the same basin, computational effort is wasted applying the local improvement procedure to find the

same exact basin. Lamarckian evolution has been shown to help alleviate this particular problem. In Lamarckian evolution, these individuals would be identical and would reproduce clones of themselves if crossed over. Thus, the local improvement procedure would not have to be applied since the children are the same as the parents. However, mutation could still cause a problem since a slight change might leave the individual in the same basin or in later generations the GA could create an individual that falls in a basin already explored. Therefore, the local improvement procedure would be reapplied to find the same basin again even with Lamarckian evolution wasting valuable computational cycles which could be used to explore other regions of the search space. In order to prevent this phenomena from occurring, adaptive memory has to be added to the GA to reduce the amount of wasted computation owing to the rediscovery of the same local minimum.

3.1 Random Linkage

The benefits (i.e., non-linearity, multi-modal) of using stochastic search methods like pure random search, simulated annealing, and genetic algorithms have been shown. However, if the function is sufficiently regular, local optimization is often relatively easy, and clustering methods may be preferred, including multi-level single linkage, and random linkage which are multistart procedures with memory.

Because the GA is working in the continuous domain, the tabu list structure used as short term memory for combinatorial problems (e.g., the list of the past allocations, last pair-wise switches, etc.) will not be an appropriate memory structure. Instead, using the concepts of global optimization (Rinnooy-Kan and Timmer, 1987), hyperspheriods are placed around the already examined points, thus marking a portion of the search space as tabu.

Random linkage (RL) is a search strategy for solving global optimization problems. The algorithm works by generating uniform random starting search locations, and applying an accept/reject criteria to each of the generated search locations. An accept decisions allows the algorithm to apply a local search to the search location resulting in the discovery of a local optimal point. The motivation behind the accept/reject decision is to not allow the algorithm to start a local search in an area too near an already found local optima. The random linkage algorithm (Locatelli and Schoen, 1999) is depicted below.

Random Linkage Algorithm

1 $k \leftarrow 0$.

2 Sample a single point X_{k+1} from the uniform distribution over Ω.

3 Start a local search from X_{k+1} with probability

$$\varphi_k(\delta_k(X_{k+1})),$$

where

$$\delta_k(x) = \min_{j=1,\ldots,k} \{\|x - X_j\| : f(X_j) \geq f(x)\}.$$

It is understood that $\delta_k(x) = \infty$ if no j exists such that $f(X_j) \geq f(x)$.

4 If the stopping criteria is satisfied, then Stop; otherwise, repeat from Step 2.

Random linkage is parameterized based upon the acceptance function φ_k; for example, the algorithm Best Start, where a local search is started only if the sample point has a functional value better than any previously seen, is obtained by letting $\varphi_k(\delta) = 0$, for all $\delta < \infty$ and $\varphi_k(\infty) = 1$; pure random search is achieved with $\varphi_k(\delta) = 0$, for all δ; and multistart with $\varphi_k(\delta) = 1$, for all δ.

Locatelli and Shoen (1999) go on to derive conditions on which allow the random linkage algorithm to have the following interesting properties:

1 Convergence to the best function value observed to the global optimum value with probability 1.

2 The probability of starting a local search decreases to 0.

3 A finite number of local searches are initiated even if the algorithm is run forever.

Random linkage provides a means for determining when a local search is appropriate; however, it relies on uniform sampling to generate new sample points. This requirement provides the algorithm the theoretical convergence properties discussed above. However, it does no insure that the algorithm is efficient for finding the global optimum of real functions. Evolutionary algorithms are very efficient at locating interesting areas of a search space, with hybrid algorithms employing local search algorithms as part of their evaluation function. However, due to an evolutionary algorithms tendency to exponentially exploit promising areas of the search space, hybrid algorithms may be wasting computational time repeating local searches.

3.2 Evolutionary Algorithms with Random Linkage

A combination of evolutionary algorithms with random linkage (EARL) is developed in an attempt to produce an evolutionary algorithm which does not repeat local searches by using RL's accept/reject criteria. Even though EARL uses the same accept/reject criteria as RL, it does not retain the theoretical convergence properties of RL due to the non-uniform sampling done by the evolutionary algorithm. Since an evolutionary algorithm is free to choose crossover and mutations, the points sampled by the algorithm can come from any possible distribution. However, to investigate the effectiveness of EARL, a series of experiments was conducted on with three real valued crossovers and five real valued mutations described in detail in (Houck et al., 1997). Random linkage was implemented in the evaluation function, where k is the number of individuals evaluated so far during the course of the simulated evolution run, and was taken from (Locatelli and Schoen, 1999) described in Equation (8.8).

Even though EARL uses the same accept/reject criteria as RL, it does not retain the theoretical convergence properties of RL due to the non-uniform sampling done by the evolutionary algorithm. Since an evolutionary algorithm is free to choose crossover and mutations, the points sampled by the algorithm can come from any possible distribution. However, to investigate the effectiveness of EARL, a series of experiments was conducted.

Tabu regions are incorporated into the genetic algorithm by keeping a list of points already generated by the GA to date; this list is allowed to grow indefinitely. Every time the genetic algorithm generates a new point, x_k, it first determines the distance, δ, to the nearest point already generated (i.e., the nearest point on the list). The newly generated point, x_k, is placed on the list only if the distance, δ, is greater than zero, which avoids placing duplicated points onto the list. The algorithm then determines whether or not a local search should be performed based on this distance. If a local search is deemed appropriate, local heuristic is run, and the resulting local minimum is returned to the GA. Otherwise, no local improvement is conducted, and the new point x_k, and the objective function value at this point are returned to the GA

To test the computational efficiency of both RL and EARL, an empirical investigation of both the quality of solutions found, as well as the computational efficiency, in terms of number of function evaluations of the algorithms was conducted. To determine when to perform a local

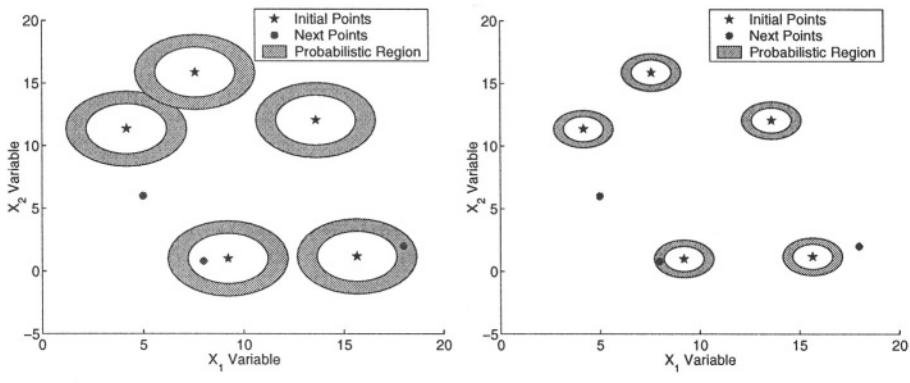
search (i.e., an appropriate distance δ), the RL algorithm and EARL use the following acceptance criteria , $\phi_k(\delta)$, (Locatelli and Schoen, 1999):

$$\varphi_k(\delta) = \begin{cases} 0 & \delta \leq 0.5\alpha_k \\ \frac{\delta - 0.5\alpha_k}{\alpha_k} & 0.5\alpha_k \leq \delta \leq 1.5\alpha_k \\ 1 & \delta \geq 1.5\alpha_k \end{cases} \tag{8.8}$$

where

$$\begin{aligned} \alpha_k &= \pi^{-1/2}\left[\Gamma\left(1+\frac{d}{2}\right)\mu(\Omega)\sigma\frac{\log k}{k}\right]^{1/d} \\ \sigma &= 4 \\ d &= \text{dimension of problem} \\ \mu(\Omega) &= \text{Lebesgue measure of search space.} \end{aligned}$$

$\phi_k(\delta)$ provides the probability of starting a local search based on the distance to the nearest point already generated, δ. The tabu regions or hyperspheres, α_k, taken from (Locatelli and Schoen, 1999), are used by to determine if a local search is necessary (See Figure 8.4).



(a) Early Generations (b) Later in the Run

Figure 8.4. Ellisphoid Hyperspheres that Represent Acceptance/Rejection Areas

All points within a radius of $\frac{1}{2}\alpha_k$ are marked tabu, where k is the total number of search points generated by the genetic algorithm to date; all points from $\frac{1}{2}\alpha_k$ to $\frac{3}{2}\alpha_k$ result in a probability of conducting a local search. This yields a gradual boundary around previously explored

points. This provides a criteria for determining whether or not to perform a local search in order to prevent starting local searches too close to points already examined.

This measure yields a series of monotonically decreasing hyperspheres (α_k) placed around each examined point (see Figure 8.4). Local search is not applied to points generated inside the ellipsoid doughnuts, while local search is always applied to points outside the doughnuts. If a point falls on a doughnut, then a local search is applied with some positive probability. Therefore, the size of the tabu regions around each of these already examined points decreases as the search progresses (i.e., this provides a short term memory similar to traditional tabu search). This also overcomes the problem of unknown minima sizes, ensuring, that smaller minima can be located as the search progresses. While at first the emphasis of the search process is on diversity (see Figure 8.4a), due to the large hyperspheres preventing much local search, as the search continues, α_k decreases so that, even though the probability of repeating a search increases, the probability of missing an unexplored local minimum decreases (see Figure 8.4b). The two sub-figures of Figure 8.4 shows five initial points with their corresponding hyperspheres and the three newly generated points ((5,6), (8,0.8), and (18,2)). If these points were generated early in the run as seen in Figure 8.4a, the local search will be applied to the first new point generated, not be applied to the second point, and applied with some probability to the third point. From Figure 8.4b which represents later in the run, the local search would be applied to the first and third point generated as well as applied with some probability to the second point. These two figures how local exploitation is increased over the run while early in the run more global exploration is emphasized.

3.3 Experimentation and Results

Two versions of a standard hybrid genetic algorithm were used to compare the results of RL and EARL, a 40% and 100% Lamarckian strategy as advocated in Section 2.2. All four search strategies were terminated when the solution value was within 1×10^{-6} of the optimal, or after one million function evaluations. Each of these test strategies were used on the same five nonlinear test problems in Section 2.2.

Using the same analysis, Tables 8.5 and 8.6 represent the summary of the SNK multiple means comparison test for solution quality and computational efficiency respectively. As Table 8.5 indicates, only random linkage was unable to consistently locate the optimal solution within the 1 million function evaluations. However for computational efficiency (as

seen in Table 8.6, EARL was the only method that was statistically in the best group for all of the the problems tested.

Table 8.5. Rankings based on Fitness

Problem Instance	RL	40%	100%	EARL
Brown2	1	1	1	1
Brown10	2	1	1	1
Brown20	2	1	1	1
Corana2	1	1	1	1
Corana10	2	1	1	1
Corana20	2	1	1	1
Griewank2	2	1	1	1
Griewank10	1	1	1	1
Griewank20	1	1	1	1
Rastrigin2	1	1	1	1
Rastrigin10	2	1	1	1
Rastrigin20	2	1	1	1
Schwefel2	1	1	1	1
Schwefel10	2	1	1	1
Schwefel20	2	1	1	1

Table 8.6. Rankings based on Number of Evaluations

Problem Instance	RL	40%	100%	EARL
Brown2	1	1	1	1
Brown10	3	2	2	1
Brown20	2	1	1	1
Corana2	2/1/1†	1	1	1
Corana10	3	2	1	1
Corana20	3	3	2	1
Griewank2	2	1	1	1
Griewank10	1	2	3	1
Griewank20	1	2	3	1
Rastrigin2	1	1	1	1
Rastrigin10	2	1	1	1
Rastrigin20	3	2	1	1
Schwefel2	1	1	2	1
Schwefel10	3	1	2	1
Schwefel20	3	2	2	1

† - Duncan Grouping / SNK Grouping / REGWF Grouping

4. Summary

Hybrid genetic algorithms use local improvement procedures as part of the evaluation of individuals. For many problems there exists a well developed, efficient search strategy for local improvement, e.g., hill-climbers for nonlinear optimization. These local search strategies compliment the global search strategy of the genetic algorithm, yielding a more efficient overall search strategy. Second, they allow the GA to be made more specific to the problem at hand since the LIP is designed to work for that problem.

It was shown that a partial or full Lamarkian evolution provided the best mix of solution quality and computational efficiency. However, evolutionary algorithms are designed to exponentially exploit promising regions of the search space. This can lead to wasted computational effort re-applying a local search in a region already explored. Random linkage, a search algorithm taken from global optimization, is designed to prevent repeated searches by using an accept/reject function which determines whether a local search is appropriate. Random linkage has powerful theoretic convergence properties which are based upon a random sampling of the search space. However, since RL uses uniform sampling to generate search points, the algorithm is unable to exploit promising regions of the search space. A combination of random linkage and evolutionary algorithms was developed by using the accept/reject criteria of random linkage in the evaluation function. This combination was developed so as to retain the exploitation of evolutionary algorithms, but by using the accept/reject criteria of RL avoiding repeated local searches due to over-exploitation. This combination strategy was shown to be computationally efficient on a series of five nonlinear test problems. Currently, the technique only works for continuous problems owing to the fact a distance has to be computed.

References

Barr, R. S., Golden, B. L., Kelly, J. P., Resende, M.G.C. and Stewart Jr., W. R. (1995) Designing and reporting on computational experiments with heuristic methods. *Journal of Heuristics,* 1(1):9–32.

Chandrasekharan, M. P. and Rajagopalan, R. (1987) Zodiac—an algorithm for concurrent formation of part families and machine cells. *International Journal of Production Reseach,* 25(6):835–850.

Chu, P. C. and Beasley, J. E. (1995) A genetic algorithm for the generalised assignment problem. Technical report, The Management School Imperial College London.

Cooper, L. (19972) The transportation-location problems. *Operations Research,* 20:94–108.

Corana, A., Marchesi, M., Martini, C. and Ridella, S. (1987) Minimizing multimodal functions of continuous variables with the "simulated annealing" algorithm. *ACM Transactions on Mathematical Software,* 13(3):262–280.

Davis, L. and Orvosh, D. (1994) Using a genetic algorithm to optimze problems with feasibility constraints. In *1994 IEEE International Symposium Evolutionary Computation,* pages 548–553, Orlando, Fl.

Davis, L. (1991) *The Handbook of Genetic Algorithms.* Van Nostrand Reingold, New York.

Glover, F. and Laguna, M. (1997) *Tabu Search.* Kluwer, Boston.

Hinton, G. E. and Nolan, S. J. (1987) How learning can guide evolution. *Complex Systems,* 1:495–502.

Hooker, J. N. (1995) Testing heuristics: We have it all wrong. *Journal of Heuristics,* 1(1):33–42. .

Houck, C. R., Joines, J. A. and Kay, M. G. (1996a) Comparison of genetic algorithms, random restart, and two-opt switching for solving large location-allocation problems. *Computers & Operations Research,* 23(6):587–596.

Houck, C. R., Joines, J. A. and Kay, M. G. (1996b) A genetic algorithm for function optimization: A Matlab implementation. Technical Report NCSU-IE Technical Report 95-09, North Carolina State University.

Houck, C. R., Joines, J. A. and Kay, M. G. (1997) Empirical investigation of the benefits of partial lamarckianism. *Evolutionary Computation,* 5(1):31–60.

SAS Institute Inc. (1990) *SAS/STAT User's Guide.* Cary, NC, 4th edition.

Joines, J. A., Culbreth, C. T. and King, R. E. (1996c) Manufacturing cell design: An integer programming model employing genetic algorithms. *IIE Transactions,* 28(1):69–85.

Joines, J. A. and Culbreth, C. T. (1999) Job sequencing and inventory control for a parallel machine problem: A hybrid-ga approach. In D. Fogel and Z. Michalewicz, editors, *Proceedings of 1999 IEEE Conference on Evolutionary Computation,* pages 579–585, Washington, DC. Institute of Electrical and Electronics Engineers.

Joines, J. A., Kay, M. G. and Houck, C. R. (2000b) Characterizing search spaces for tabu search and including adaptive memory into a genetic algorithm. *Journal of the Chinesse Institute of Industrial Engineers.*

Joines, J. A., Kay, M. G., King, R. E. and Culbreth, C. T. (2000a) A hybrid genetic algorithm for manufacturing cell design. *Journal of the Chinesse Institute of Industrial Engineers.*

Joines, J. A., King, R. E. and Culbreth, C. T. (1994) A comprehensive review of manufacturing cell design. Technical Report NCSU-IE Technical Report 94-20, North Carolina State University.

Joines, J. A., King, R. E. and Culbreth, C. T. (1996d) Moving beyond the parts incidence matrix: Alternative routings and operations for the cell formation problem. *International Journal of Engineering Design and Automation,* to appear.

Joines, J. A. (1996) *Hybrid Genetic Search for Manufacturing Cell Design.* Ph.d. thesis, North Carolina State University, Raleigh, NC, December 1996.

Lawrence, C., Zhou, J. L. and Tits, A. L. (1994) User's guide for cfsqp version 2.4: A c code for solving (large scale) constrained nonlinear (minimax) optimization problems. TR 94016rl, University of Maryland, Electrical Engineering Dept and Institute for Systems Research.

Locatelli, M. and Schoen, F. (1999) Random linkage: A family of acceptance/rejection algorithms for global optimization. *Mathematical Programming,* 85(2):379–396.

Love, R. F. and Juel, H. (1982) Properties and solution methods for large location-allocation problems with rectangular distances. *Journal Operations Research Society,* 33:443–452.

Michalewicz, Z. (1996) *Genetic Algorithms + Data Structures = Evolution Programs.* AI Series. Springer-Verlag, New York, 3rd edition.

More, J. J., Garbow, B. and Hillstrom, K. (1981) Testing unconstrained optimization software. *ACM Transactions on Mathematical Software,* 7(1):17–41.

Nakano, R. (1991) Conventional genetic algorithm for job shop problems. In *Proc. of the 4th ICGA,* pages 474–479, San Mateo, CA.

Ng, S. (1993) Worst-case analysis of an algorithm for cellular manufacturing. *European Journal of Operational Research,* 69(3):384–398.

Renders, J. -M. and Flasse, S. (1996) Hybrid methods using genetic algorithms for global optimization. *IEEE Transactions on Systems, Man, and Cybernetics Part B:,* 26(2):243–258.

Rinnooy-Kan, A. and Timmer, G. (1987) Stochastic global optimization methods part I. *Mathematical Programming,* 39(l):27–56.

Whitley, D., Gordon, S. and Mathias, K. (1994) Larmarckian evolution, the Baldwin effect and function opimization. In Y. Davidor, H.P. Schwefel, and R. Manner, editors, *Parallel Problem Solving from Nature-PPSN III,* pages 6–15. Springer-Verlag.

Wolpert, D. H. and Macready, W. G. (1997) No free lunch theorems for optimization. *IEEE Transactioins on Evolutionary Computation,* l(l):53–66.

Chapter 9

USING EVOLUTIONARY ALGORITHMS TO SOLVE PROBLEMS BY COMBINING CHOICES OF HEURISTICS

Peter Ross and
Emma Hart

Abstract

Potential users of evolutionary algorithms (EAs) are often deterred by the 'black box' nature of many of the available examples. Typically an evolutionary algorithm is designed to search a problem's solution space directly, and the user simply waits for some stopping criterion to take effect. However, the user usually gets no guarantees about the quality of the fittest solution at that point. It is unsurprising, therefore, that users may choose to use some simpler, cheaper heuristic method whose performance is better understood and faster even though it may well deliver poorer results. Commercial users in particular often have good reason to be nervous about using EAs in situations in which their business is likely to be judged on the quality of the EA's result. However, there are ways in which they can still use EAs to very good effect. This paper discusses one such way, namely using an EA to choose which heuristics to apply at each stage in some sequential decision process. If the available heuristics are individually acceptable, then a combination of them is going to produce a better quality result than any of them individually would. This can be guaranteed by the simple device of seeding the initial population with chromosomes that employ just one heuristic throughout. The paper describes some examples of this approach and discusses possible developments of the idea.

Keywords: Heuristics, hyper-heuristics, scheduling, timetabling

1. Introduction

Software vendors and academics often extol the virtues of evolutionary algorithms as a means of exploring very large search spaces. Customers

are often justifiably skeptical, particularly if they only want something that they know works well enough and quickly enough, rather than something which strives at great effort to find very high quality answers. There are many good reasons for choosing a more traditional algorithm over an EA, for example:

- that algorithm has known performance bounds;
- the algorithm is faster than any population-based method;
- the algorithm has a good track record in commercial use;
- there is less personal or organisational risk in 'following the crowd' than in opting for unfamiliar technology.

By contrast, the EA community has sometimes been guilty of looking inward and failing to make proper comparisons with non-evolutionary methods, or a proper case for using an EA at all. To be fair, it can be very tricky to run proper comparisons; benchmark problem sets may be unrealistic or unrepresentative of the user's own problems, and algorithms that contain stochastic elements will produce a spectrum of results so that it can be hard to assess the true cost of obtaining a minimally acceptable result by repeated runs.

In some cases it is possible for a user to tell whether an EA has completely solved a problem. For example, penalty-based EAs often calculate fitness by assigning a weight $w_i > 0$ to constraint violations of type i, and if there are p_i of them the fitness is defined to be

$$f = 1/(1 + \sum_i w_i p_i)$$

In this case, any chromosome with fitness 1.0 represents a perfect solution. However, if the EA only finds chromosomes with lower fitness, the user is left wondering whether the problem as given is unsolvable or whether the EA is simply unable to discover any of the perfect solutions, either because the EA is underpowered or because the choice of weights was flawed. In many other cases the fitness function does not even have a known maximum, and the EA essentially functions as a fitness-improving black box.

On the other hand, familiar heuristics may have their drawbacks too. Consider the bin-packing problem. In its simplest, idealised form there are many bins of known, identical capacity and a number of items of various sizes to be packed into as few bins as possible. The *First Fit Decreasing* algorithm orders the items by size, largest first, and simply puts each into the first bin in which it will fit, starting a new bin whenever

necessary. Many companies some variation of this for packing tasks; it is simple, fast and comprehensible. Johnson, in (Johnson, 1972), shows that this algorithm uses no more than $11b/9 + 1$ bins where b is the optimal number, and that no other such algorithm can hope to do better than this. However, consider the problem (attributed to R.L.Graham of AT&T Bell Labs, (Hoffman, 1998)) of packing these items into bins of size 524:

442	252	127	106	37	10	10
252	252	127	106	37	10	9
252	252	127	85	12	10	9
252	127	106	84	12	10	
252	127	106	46	12	10	

The result uses seven bins:

Bin 1:	442	46	12	12	12	
Bin 2:	252	252	10	10		
Bin 3:	252	252	10	10		
Bin 4:	252	252	10	10		
Bin 5:	252	127	127	9	9	
Bin 6:	127	127	127	106	37	
Bin 7:	106	106	106	85	84	37

but if the single item of size 46 is *deleted,* the algorithm needs eight bins instead:

Bin 1:	442	37	37					
Bin 2:	252	252	12					
Bin 3:	252	252	12					
Bin 4:	252	252	12					
Bin 5:	252	127	127	10				
Bin 6:	127	127	127	106	10	10	10	
Bin 7:	106	106	106	85	84	10	10	9
Bin 8:	9							

As this shows, the heuristic can perform in an unexpected and undesirable way that is hard to anticipate: solving a sub-problem produces a worse answer than solving the whole problem. There are of course more sophisticated bin-packing algorithms than this, see (Coffman et al., 1997) for a survey.

More generally, there are huge numbers of algorithms for finding approximate solutions to OR-type problems, and in many cases there are theoretical results about how well or badly they can perform; see (Ausiello et al., 1999) for an introduction and a compendium of known results for over 200 types of problem. However, experience suggests that, given a heuristic or approximate algorithm, it is often possible to devise examples that exploit its weaknesses in some way so that the heuristic produces a relatively poor answer. Rather than merely accepting this risk as the price of comparative simplicity, it is often possible to combine choices of heuristic so as to produce better overall performance than any one heuristic.

This paper describes examples of how EAs can be used to combine choices of heuristics. The next section briefly considers pre-cursors of this idea, in which a parameterised algorithm is used to solve some problem and a GA is used to find good choices for those parameters. The following ssections then describe examples in which a GA is used to select choices of heuristic as part of an algorithm; these examples are all drawn from the general area of scheduling and timetabling, and illustrate variations on the theme. Finally there is a brief discussion of how the general idea might be developed further.

2. GAs and parameterised algorithms

There are many examples in the literature in which a GA is used to find a good set of parameters for some parameterised algorithm. Two examples are briefly described below. This represents a useful step forward from the naive approach of directly encoding a potential solution in a chromosome. Direct encoding has the potential drawback that, for complex real-world problems, the chromosome may need to be very long. There is then a real danger that some parts of the finally-produced solution may be due only to genetic drift, rather than because those parts have been selected on the basis of their contribution to overall fitness. Working with a parameterisation of some kind can help to keep chromosome length more under control.

The first example is a commercially very successful example, in which a GA is used to configure the search for the best way to set up a robot that mounts chpos and other parts onto computer boards. The second example is dran from the world of futures trading.

2.1 The Philips FCM robot

Fig. 9.1 shows a plan view of part of the 'Fast Component Mounter' robot sold by Philips for placing components such as chips onto surface-mount boards.

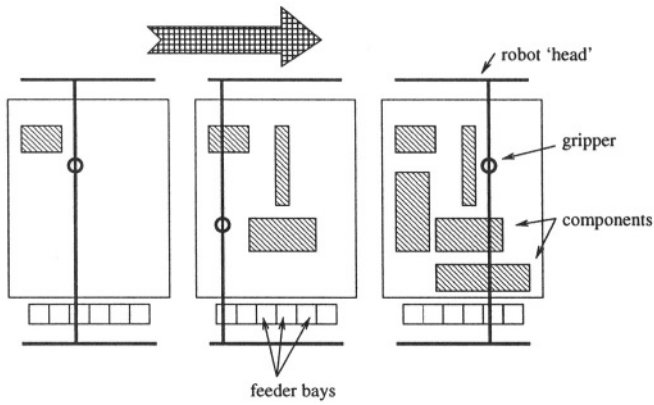


Figure 9.1. Philips FCM robot

This is an idealised picture, for clarity. In practice there are 16 robot heads and, unlike in the figure, each robot head can only reach a small part (a vertical slice, narrower than shown) of a board. Boards advance in lock-step beneath the heads. Each robot head has a suction gripper, of one of various sizes; the grippers are not interchangeable during a run. Components arrive via parts feeders, and a feeder for large parts such as CPU chips can occupy up to three feeder bays. Each robot head runs a cycle of picking up a part from one of the feeders associated with it, moving the part to an alignment jig in order to get the part into a known orientation, and then moving to the appropriate location on the current board and inserting the part there.

The problem therefore is to decide: for each head, which type of gripper it has; for each head, which types of feeder it has and where they are placed in the feeder bays; for each head, what alignment tooling it has; for each feeder, which parts it feeds (a feeder need not feed just one type of part, but the parts it feeds must be appropriate for the size of feeder). The objective is to minimise the time taken to assemble each board. It is not very practicable to treat the subtasks as separate optimisation problems, since there is no good way to evaluate a potential solution to any one subtask in isolation. Attempts to develop approximate evaluation functions have not been good enough to be useful.

The solution that was adopted was to use a set of numerically parameterised greedy algorithms for each of the subtasks. Each chromosome in an EA contained one set of all the parameters for all the algorithms, and was evaluated by running the algorithms and then timing the performance of the resulting configuration, by straightforward simulation. The parameters involved are essentially weights representing 'desirability'. In gripper assignment, a weight represents the desirability of using a certain kind of gripper; in feeder assignment, there is weight representing the desirabiluty of assigning a certain feeder to a given head, that already has an assigned gripper type from the previous stage of the algorithm; and so on. The results have been extremely successful; despite extensive efforts, it has not been possible to improve on the solutions produced by the EA, either by computer or human means. Moreover, it has not been necessary to tweak the EA for different sorts of board being manufactured. Philips supply the EA with the robot, the particular EA used is Eshelman's very successful CHC algorithm. Further details can be found in (Schaffer and Eshelman, 1996).

This approach represents an interesting alternative to what the textbooks often seem to recommend, namely some kind of direct attack on a combinatorial optimisation task using, say, a permutation-based encoding of solution ingredients.

2.2 Futures Trading: When To Buy

In the second example, see (Katz and McCormick, 1997) and (Katz and McCormick, 2000), a GA is used to find a rule for deciding when to enter a certain futures market, that is, at what moment to buy. The idea is to find a combination of three rules, each chosen from a set of ten possible parameterised rules, and the moment of entry is the moment when all three rules return the value 'true'.

Market information arrives in a continual stream of data-points, called bars. One rule might say: "return true if the closing price n_1 bars ago exceeds the closing price n_2 bars ago by at least an amount t". Another might say: "return true if the current price is above or below (according to the value a binary variable v) a simple linear moving average of prices over the last b bars". There is a maximum of three parameters per rule, possibly fewer as the second example rule suggests, and so the chromosome consists of three block of four numbers, one block per rule. Within a block, the first is an integer specifying which of the ten rules the block refers to; the other three numbers are rule parameters.

As reported in (Katz and McCormick, 2000), this GA is capable of finding reasonably good-quality market entry strategies, even though for

the sake of good science a fixed and possibly sub-optimal market exit strategy is used. Fitness is judged by overall profitability over a fairly long period. Note that the chromosome is very short, and that the three blocks need not refer to different rules – the entry condition may therefore be determined by two or even just one of the ten parameterised rules under consideration.

2.3 Discussion: parameterised algorithms

In both of the above examples, real-valued parameters are used to flesh out some pre-determined algorithm. There are some clear advantages over using a direct encoding, notably that the chromosome used can be very much shorter than in a direct encoding and that there is at least some possibility of analysing why the discovered solution works. There is also at least the possibility of demonstrating that the GA has improved upon some previously known algorithm, by choosing the parameters that would represent that previously known algorithm and injecting the corresponding chromosome into the initial population.

Rather than using real-number parameterisations, the following sections describe examples drawn from our own experience in which the idea is to use a GA to combine choices of heuristics that are already known and used in the relevant application area. We refer to this as a *hyper-heuristic* approach.

3. Job Shop Scheduling

The job-shop scheduling problem (JSSP) is well known as one of the most difficult NP-hard problems that cannot be solved in polynomial time. It has received a great deal of attention from the evolutionary computing community because the general problem can serve as a simplified model of many practical scheduling problems. The general job-shop problem assumes that j jobs must be processed on m machines and each job follows a predefined job-specific route through the machines, and has a fixed processing time p_{jm} on each machine. All operations are non-preemptive. In the *static* version of the problem all jobs are available for processing at time 0, and the academic goal is usually to minimise the *makespan,* the time taken for all jobs to complete. In the *dynamic* version, each job has a release date r_j at which it becomes available for processing, and a due-date d_j by which time it must complete. The importance of each job is described by a weight w_j. The dynamic version of the problem has much more relevance to real life problems and hence is of more benefit to study. Two types of dynamic JSP can be considered — the *deterministic* problem in which all release dates and due-dates

are known at the start of the problem, and the *stochastic* problem, in which jobs arrive at some unknown time in the future.

Many approximate and heuristic-based methods have been proposed to attempt to find near-optimal solutions to large problems in realistic time-scales, a large number of which include the use of evolutionary algorithms. A wide variety of EA techniques have been proposed, using many different representations and operators, and relating to a broad spectrum of problems, e.g. (Bruns, 1997). All EA methods encounter the problem of how to represent a potential schedule as a chromosome. Early work in the field yielded two extremes of possible approaches to representing schedules. Nakano and Yamada in (Nakano and Yamada, 1991) used a binary representation to encode schedules for benchmark job-shop problems. A complex effort was required to design such an encoding, and its use in a GA context needed specialised repair operators to retain the ability to decode chromosomes as feasible schedules. The other extreme, for example see (Bruns, 1993), is a direct encoding of a schedule, in which an encoded schedule was a direct representation of the schedule itself, with data structures and attributes designed to mimic the real schedule. Hence no decoding of a chromosome was necessary but the genetic operators needed to use much domain and constraint knowledge to appropriately mutate schedules, and to maintain feasible schedules. Thus the resulting algorithms were extremely problem-specific and could not be generalised to other types of problems or even other instances of the problem type under consideration. An intermediate approach uses an indirect encoding in which a schedule building algorithm is combined with genetic search through the space of potential inputs to the schedule builder, for example see (Fang et al., 1993). The resulting schedules are often fragile however, in that slight perturbations lead to large changes in the schedule, and the chromosome representation gives little clue to how and why the schedule was built, and is therefore difficult to modify by hand.

However, a large number of fast and cheap heuristics are available for the JSSP problem, accumulated over many years of research by the Operations Research community. A sample of the available heuristics is given in table 9.1 — each heuristic indicates which job should be placed next in the schedule. For example, heuristic WSPT chooses the job with the smallest weighted processing time, WLWKR chooses the job with the least work remaining etc. A handful of EA applications, loosely defined as 'Heuristic Combination Methods' (HCM) have attempted to take advantage of these heuristics by using an EA to search for successful sequences in which to apply these heuristics in order to build a schedule. There is no reason to rely on a single heuristic when building a schedule

Rule	Description
WSPT	Weighted shortest processing time
LWKR	Weighted Least Work Remaining
WTWORK	Weighted Total Work
EGD	Earliest Global Due Date
EOD	Earliest Operational Due Date
EMOD	Earliest Modified Operational Due Date
MST	Modified Slack Time
SOP	Slack per Operation
POPNR	Lowest ratio of processing time of imminent operation to weighted value of remaining operations
PSOP	Weighted smallest sum of (next processing time + SOP)
PWKR	Weighted smallest ratio of processing time to work remaining
RND	Choose a Random operation from those operations that cannot be chosen by another heuristic

Table 9.1. Heuristics Used for Dynamic Job-Shop Problems

— indeed, it seems obvious that varying the choice of heuristic according to the particular stage of the scheduling process or to the size of the job being processed makes sense. It is difficult of find some principled method of making the choice of heuristic at each decision point, but an EA can at least be used to search for successful combinations of such heuristics.

Norenkov, see (Norenkov and Goodman, 1997), describes an EA in which schedule synthesis is considered in two phases; first the jobs are ordered by choosing one of a set of possible ordering rules, $\{A\}$, then jobs are assigned to machines according to one of a set of assignment rules $\{B\}$. A heuristic H is generated by combining a rule from set A with one for set B. The GA described searches for the optimal sequence of application of the set of possible heuristics. Using this algorithm, they obtained promising results on a set of flow-shop scheduling problems.

Dorndorf and Pesch, see (Dorndorf and Pesch, 1995), proposed a 'priority-rule based' genetic algorithm, which uses some of the heuristics defined in table 9.1. The EA they describe searches for the optimal sequence of heuristics to use in which to choose jobs to place into the schedule. However, these heuristics only specify *which* job should be placed next in the schedule, they do not indicate *where* that job should be placed. Dorndorf and Pesch rely on the *Giffler and Thompson* algo-

rithm ((*G* + *T*) in order to place jobs in the schedule. This algorithm was proposed in 1960 and generates a solution in which it is guaranteed that the resulting schedule contains no idle time and none of the operations can be finished earlier without delaying another operation. This is known as an active schedule, and it can be proved that the optimum solution lies within the region of active schedules. An outline of the *G* + *T* algorithm is given in figure 9.2. Dorndorf and Pesch's algorithm uses each heuristic to select an operation in step S4. They tested this algorithm on a number of static job-shop benchmark problems, using makespan as the objective, obtaining results which have since been superseded by other methods, for example see (Vaessens et al., 1996).

We proposed an extension to this method in (Hart and Ross, 1998; Hart and Ross, 2000) for tackling dynamic job-shop scheduling problems. It was stated above that the optimal schedule is guaranteed to fall within the space of active schedules. However, a subset of this space is that of *non-delay* (ND) schedules in which operations are placed into the schedule such that machine idle time is minimized. No machine is kept idle if some operation can be processed. The method by which members of this subset of schedules is generated is given in figure 9.3. Therefore, for some problems, it may be sufficient to search only the space of non-delay schedules, which is much smaller than the set of all active schedules. However, there is no way of knowing *a priori* whether the optimal schedule lies in this subset or not, therefore only searching the non-delay subset could lead to a sub-optimal solution. Therefore the method we propose extends the HCM by evolving both the *method* to generate a set of schedulable operations (set G in figures 9.2 and 9.3, and the heuristic which then chooses an operation from this set. Each gene in the chromosome encodes a pair (*Method, Heuristic*), and there is one gene for every decision point, i.e. every operation that must be scheduled. The representation guarantees a feasible solution, and straightforward recombination operators can be used which always produce feasible offspring. We refer to this method as *HGA — heuristically-guided GA.*

3.1 Experiments

A series of experiments investigated the performance of HGA on a set of 12 benchmark problems which varied in size from (10x3) to (50x8). Each experiment was run using 4 different objective values for measuring the quality of the schedule, resulting in 48 different experiments. The objective values are defined in table 9.2. 12 heuristics were available to the GA for selecting an operation. Other experiments investigated the effect of fixing the choice of method to either ND or G+T, and

1. Calculate the set C of all operations that can be scheduled next
2. Calculate the completion time of all operations in C, and let m^* equal the machine on which the minimum completion time t is achieved.
3. Let G denote the conflict set of operations on machine m^* - this is the set of operations in C which take place on m^*, and whose start time is less than t.
4. Select an operation from G to schedule
5. Delete the chosen operation from C and return to step 1.

Figure 9.2. Giffler and Thompson Algorithm

1. Calculate the set C of all operations that can be scheduled next
2. Calculate the starting time of each operation in C and let G equal the subset of operations that can start earliest
3. Select an operation from G to schedule
4. Delete the chosen operation from C and return to step 1.

Figure 9.3. Non Delay Algorithm

also the effect of including the RND heuristic which randomly selects an operation. Results were compared to those obtained by using Priority-Rules alone, and to two other evolutionary approaches; that of Fang (1994), and a direct approach recently reported by Lin *et al.* (1997), using a parallel GA they refer to as *PGA,* in which the chromosome directly encoded a GANNT chart representing the schedule. In each case we compared the best result found in 10 repeated experiments, to those results reported by (Lin et al., 1997) and (Fang, 1994) (who also report best of 10).

Experimental parameters. We use a parallel GA, with 5 sub-populations of size 50 arranged in a ring, with migration of one chromosome from one population to the next occurring every 5 generations, in order to be consistent with the method used by Lin. The length of a chromosome is equal to the number of operations to be scheduled. Uniform crossover is used, and crossover always takes place between gene pairs, so that an (M, H) schema is never destroyed by crossover. A mutation operator mutates each heuristic in the genome to another randomly chosen heuristic with probability $p = 0.01$. For each heuristic mutated, the corresponding method allele is mutated with probability

Objective Function		Objective	Normalised Definition
ETwt	Weighted Earliness + Tardiness	Minimize	$\left(\sum_j w_j(E_j + T_j)\right) / \left(\sum_j w_j P_j\right)$
Fwt	Weighted Flowtime	Minimize	$\left(\sum_j w_j(C_j - r_j)\right) / \left(\sum_j w_j P_j\right)$
Twt	WEighted Tardiness	Minimize	$\left(\sum_j w_j T_j\right) / \left(\sum_j w_j P_j\right)$
Lwt	Weighted Lateness	Minimize	$\left(\sum_j w_j L_j\right) / \left(\sum_j w_j P_j\right)$

Table 9.2. Objective Function Definitions. If the completetion time of job j is C_j and the due-date d_j, then the lateness L_j is $(C_j - d_j)$, the earliness E_j is $max(-L_j, 0)$ and the tardiness T_j as $max(L_j, 0)$. P_j represents the total processing time of job j

0.5. We use a generational reproduction strategy, with rank selection. All experiments are run for 1000 generations, to allow a fair comparison with the experiments of Lin *et al.* (1997) and Fang (1994).

3.2 GA Parameters

Full results are given in (Hart and Ross, 1998). Here we summarise the results by reporting the relative performance of HGA in each of the 48 experiments compared to the other methods. This is shown in table 9.3. The table shows in how many of the 48 experiments HGA outperformed, equalled or underperformed the other methods. The table shows that clearly HGA outperforms both Priority Rules and Fang's GA.. The performance of *HGA* is less robust when compared to those results recently published by Lin — it outperforms Lin's *PGA* in 18 cases, equals the results in a further 9 cases, and it is beaten in the remaining 21 cases. However, Vasquez and Whitley (M.Vasquez and D.Whitley, 2000) have since noted that they were unable to replicate the results reported by Lin when re-implementing the algorithm, in a paper on comparison of GAs for dynamic scheduling problems.

For all objectives, evolving the choice of scheduling method is beneficial. Comparison of HGA to the simplified HGA in which the choice of method is restricted to G&T only (called *HGA*(*G&T*)) shows that in only 3 cases does the evolution of method hinder the search and decrease performance. For the remaining cases, *HGA* produces better results than *HGA*(*G&T*) in 25 cases, and equivalent results for the other 20 cases. Comparison to Non-Delay shows 4 cases where better results are achieved using Non-Delay only. *HGA* produces better results than *HGA*(*ND*) on 40 cases, and equivalent results on the remaining 4 cases.

	Priority Rules	Fang	Lin
Better Performance	44	40	18
Equal Performance	2	5	9
Worse Performance	2	3	21

Table 9.3. The table compares the performance of HGA to 3 other methods, and shows in how many of the 48 experiments it beat the other methods, in how many experiments it equalled the other methods, and in how many experiments it was outperformed.

The decision as to whether or not to include the RND heuristic is less clear. Comparison of the results shows that in just over half the cases (28) no difference between results is observed. From the remaining 20 cases, better results are obtained in 9 cases by including the RND allele, and worse results on 11 cases. Hence, although not including the RND allele may prevent some operations from being chosen at some points in the schedule, this appears to slightly outweigh the possible disadvantages of including it, i.e. that the population may be unstable from one generation to the next.

3.3 Conclusions about JSSP

This new GA which evolves a combination of scheduling algorithm and heuristic choice to be used at each stage of the scheduling process performed very well compared to other published results on benchmark problems. In particular, evolving the choice of scheduling algorithm appears to be highly beneficial. The representation is straightforward, and can be used with standard recombination operators, which always produce feasible solutions. The results are promising across a range of problems and objectives when compared to others recently published. The GA could perhaps be tuned further with respect to individual objectives by using a specialised set of heuristics for the objective in question, rather than the general set used here.

4. Scheduling chicken catching

We have also applied a hyper-heuristic approach to solving a real-life scheduling problem faced by a local company which collects up to 1.3 million live chickens from farms geographically distributed across Scotland and processes them at two factories. The scheduling must be accomplished using a fixed resource of 'catching squads' and vehicles,

in such a way that the factories are supplied with birds at a constant rate and that all daily orders for birds are met. The problem is highly constrained, with regard to regulations governing working conditions for the workforce, rules concerning chicken welfare and to the number of fixed resources available. For example, there are strict regulations governing the time that birds can be kept in vehicles before being processed, hence the factory must operate almost on a just-in-time basis. Union rules also govern the minimum and maximum amount of work that any catching squad can do both daily and weekly, and there are also regulations concerning the number of hours that the vehicle drivers can drive without a break. There are some restrictions also on the order in which certain farms can be visited by catching squads in an attempt to prevent the spread of disease between farms. Several practical considerations must also be taken into account — for example, it is sensible for a catching squad to work within a geographically constrained area, rather than travelling backwards and forwards across the country, and it is not practical for a squad to visit a farm to collect less than a complete vehicle load of birds.

Scheduling the daily orders is a full time job for one employee who performs the process manually. Whilst it is of interest to the factory to minimise the resources required, the overriding aim is to produce *practical* daily schedules that meet the incoming orders. The problem breaks down into two sub-problems. First, the daily orders (which consist of large numbers of birds) must be split into smaller tasks that can be equitably and practically distributed amongst the catching squads. Second, these tasks must actually be distributed amongst the catching squads. The task is akin to bin-packing, if a squad is considered as a bin, however standard heuristics for performing bin-packing cannot be applied due to the constraints existing between items in each 'bin' and between bins. The manual scheduler accomplishes the task through application of a number of rules of thumb, acquired over years of experience. The heuristics are applied through intuition, and it is difficult to capture in logic the reasoning behind the method of application. Therefore, in order to try and automate the task, we need to find a method of knowing which of these heuristics to apply and when in order to produce a practical schedule. A hyper-heuristic approach can again be used; a hyper-heuristic must choose when and where to apply each of the lower-level heuristics currently used manually. Again, we use an evolutionary algorithm to evolve the sequence of application of the heuristics.

4.1 The approach

The chromosome is represented as a matrix with three rows, and n columns, one for each order to be processed. The first row represents the order in which the daily orders should be scheduled. The second row represent the heuristics that should be used to split each order into smaller sub-tasks, and the final row encodes the heuristics to use to assign the sub-tasks to squads. Therefore each vertical column represents a triple {*order, split – heuristic, assign – heuristic*}, defining how an order is split and assigned. The matrix can effectively be considered as a list of instructions for building a solution to a particular problem — however, the instruction set is unique to each different problem solved and hence tailored to solve any peculiarities of that problem. The approach is similar to that taken by Norenkov and Goodman (1997), however in their work they combined the heuristics for ordering and assigning operations into new single heuristics; the GA then searched the space of the new heuristics. In our approach, we evolve the choice separately.

Ten different heuristics are used to split the orders into smaller sub-tasks. Seven heuristics are used to then assign these orders. The performance of the GA was tested on the real data provided from eight different days at the factory. In each case, we were able to find a practical solution to the scheduling problem and the schedules appeared similar to those currently being created by hand, suggesting that the hyper-heuristic was indeed modelling the knowledge of the human scheduler. To illustrate the necessity of applying different heuristics to different instances of the same problem class, consider tables 9.4, 9.5. These tables show for each problem, the frequency with which each heuristic was chosen in solving the problem. In the case of the *split* heuristics, the distribution is clearly not random, showing that each individual problem required a different combination of heuristics in order to solve it. The distribution in frequency for application of the *assign* heuristics however appears more uniform; the distribution does not deviate significantly from the 14% frequency that would be expected for each heuristic if the choice was completely random.

Clearly the hyper-heuristic approach has been successful in this case. We have shown that an individual heuristic or algorithm would have been incapable of solving all the problems whereas our our hyper-heuristic approach produces successful results. Furthermore the company are able to understand the method in which each schedule was built — this makes the task of altering schedules due to any last minute changes in orders or factory circumstances more straightforward. The method can be generalised to other real-world problems which often contain a large amount

Test Case	% heuristic applied to split an order									
	A	B	C	D	E	F	G	H	I	J
1	**84**	3	11	1	1	0	1	0	0	0
2	**78**	6	2	1	14	0	0	0	0	0
3	**44**	6	38	0	0	0	10	0	1	1
4	35	4	**43**	1	9	1	3	0	3	3
5	**50**	10	19	1	8	0	0	10	0	0
6	**79**	12	9	0	0	0	0	0	0	0
7	**47**	29	14	0	0	0	6	3	1	0
8	**43**	33	3	9	1	0	2	1	0	0

Table 9.4. Frequency of Application of Splitting Heuristic

of domain knowledge that is generally complex and often difficult to formulate into exact procedures for solving problems. By encapsulating the knowledge in the form of heuristics, and using an EA to find the optimal combination of these heuristics we considerably simplify the task. The key advantage however lies in the flexibility that such a system provides — real problems do not follow an exact pattern from day to day, and each schedule may require a subtly different approach. Using a hyper-heuristic approach, a unique schedule builder is built each time a problem is solved, which takes account of the individual characteristics of each problem. This results in a much more robust system that copes well with variations within the data and hence can be used by the factory as a reliable tool.

5. Timetabling

Yet another problem to which we have applied the hyper-heuristic approach is that of examination timetabling, for which many heuristics are available due to the similarity between timetabling and graph-colouring problems. In an exam timetabling problem there are some hard constraints: nobody can be in two places at once; room capacities cannot be exceeded even though it may be practicable to schedule two or more different exams in the same room at the same time; there may be a hard limit of the number of available timeslots; there may be others, such as restrictions on when certain exams can happen. Typically there will also be some soft constraints, such as a wish to minimise the number of instances in which some student takes two exams close together ('near-

Test Case	% heuristic being applied to assign an or der						
	A	B	C	D	E	F	G
1	15	**19**	12	15	14	13	11
2	8	14	13	13	13	**28**	12
3	16	16	10	13	**18**	16	11
4	21	16	4	**25**	13	10	11
5	8	22	15	9	**23**	16	7
6	14	**18**	18	12	13	13	14
7	17	**21**	8	14	18	11	11
8	**26**	16	10	15	11	10	12

Table 9.5. Frequency of Application of Assignment Heuristic

clashes'). Unfortunately authors can differ as to what this concept of a near-clash means. For some, only adjacent exams are near-clashes; for others, small gaps of 1-3 free exam periods between exams will count, with weightings used to express the fact that smaller gaps are less desirable than large ones. For yet others, near-clashes can only occur between exams within the same day, so that a student who takes an exam at the end of one day and another at the start of the next does not suffer a near-clash. However, the basic problem of ensuring that nobody needs to be in two places at once can be treated as a graph-colouring problem, in which nodes represent exams, edges represent the constraint that the joined nodes must not be in the same time-slot, and the task is to colour the nodes so that no two nodes joined by an edge are the same colour. Colours correspond to timeslots. There are many good approximation methods for the graph-colouring problem, e.g. (Culberson, 1992; Petford and Welsh, 1989). Perhaps the best-known is the DSATUR algorithm (see Brelaz, 1979) which sequentially assigns colours to nodes. Nodes are ordered according to the number of colours already assigned to adjacent nodes, with tie-breaking by node degree. This algorithm an be modified to do a modestly good job of solving exam timetable problems, including soft constraints (Ross et al., 1997). EAs have also been heavily investigated as tools for solving exam timetabling problems, see eg (Burke and Ross, 1996; Carter, 1998).

Previous studies, e.g. see (Carter et al., 1995; Minton et al., 1992), have shown that some heuristics are good at solving certain instances of timetabling problem, but it is not clear which of the available heuristics should be used for solving a given set of problems, or even for solving a

particular instance of a problem. Some strategies appear in general to work well for some sets of problems, but occasionally an instance of the problem demands a different approach. This may be due to particular features of the problem which determine whether or not some heuristic may be useful.

As an example, consider a problem in which there are a number of very large exams to schedule. It may be necessary to attempt to pack these exams in a manner that tries to leave many timeslots free, ie try to pack those large ones into few timeslots, rather than spreading the exams out, because otherwise one may run out of timeslots. Therefore, a heuristic which paid attention to the packing question before considering other constraints may well work better.

5.1 A hyper-heuristic timetabler

As reported in (Terashima-Marin et al., 1999), it makes sense to build a timetable sequentially. The idea is to divide the task into two phases. In the first phase, a certain strategy is chosen, and also a heuristic for choosing which exam to consider next, and a heuristic for deciding which timeslot to put it in. After a certain condition is met, the second phase starts; this also applies a certain strategy and pair of heuristics. The GA chromosome therefore encodes: (a) choices of strategy and heuristics used in phase 1; (b) when to switch: either after placing α exams, or after placing the largest N% of exams; (c) choices of strategy and heuristics used in phase 2. The strategies considered are: (i) backtracking, in which a failure to place an exam causes the search to reconsider earlier decisions, in reverse chronological order; (ii) forward checking, in which placing an exam triggers lookahead to weed out future choices that would be incompatible with it (iii) a simple Brelaz-like algorithm in which neither lookahead nor backtracking occurs. The exam-choosing heuristics used are

- By Number of instances, ie number of constraining exams, weighted by number of people involved
- By Number of edges, ie constraining exams
- By Number of people taking the exam.
- By sum of the above three numbers
- At Random.
- By Number of instances on the nodes using the available colours.

- By Number of constraining exams on the nodes using the non-available colours.
- By Number of people on the nodes using the available colours.
- Random (pre-established). At start this random ordering between each pair of nodes is established so that when the same situation is encountered a consistent decision is made.

The slot-choosing heuristics used are as follows, bearing in mind the assumption that there are three slots per day:

- By considering the available timeslots in the order: 1, 4, 7, ...3, 6, 8, ...2, 5, 8, ...
- By considering the available timeslots in the order: 1, 3, 4, 6, 7, 9, ...2, 5, 8, ...
- By considering the available timeslots in the order: 1, n, 3, $n-2$, ...2, 5, 8, ...
- By considering the available timeslots in the order: 1, 2, 3, ..., n.
- By increasingly ordering the timeslots by the number of incoming instances from the nodes in adjacent timeslots.
- By increasingly ordering the timeslots by the number of incoming edges from the nodes in adjacent timeslots.
- By increasingly ordering the timeslots by the number of people involved in exams in adjacent timeslots.
- By increasingly ordering the timeslots by the sum of incoming instances, edges, and people from the nodes in adjacent timeslots.
- By increasingly ordering the timeslots by the number of exams using each of them.
- At random.

The GA was applied to problems from Carter's set of large exam timetabling problems at `ftp://ftp.mie.utoronto.ca/pub/carter/testprob/`. Table 9.6 gives some information about these problems.

Experiments were carried out using a GA with 5 populations each of size 25, with migration between them every 10 generations, modified tournament selection of size 5 and two-point crossover. There were five runs per problem. Table 9.7 shows the results. In this table the entries

Problem	Exams	Students	Max Sz Exam	Edges	Max Slots	Seats
HECS92	81	2823	634	1363	18	
STAF83	139	611	237	1381		
YORF83	181	941	175	4706		
UTES92	184	2750	482	1430		
EARF83	190	1125	232	4793		
TRES92	261	4360	407	6131	35	655
LSEF91	381	2726	382	4531		
KFUS93	461	5349	1280	5893	20	1955
RYES93	486	11483	943	8872		
CARF92	543	18419	1566	20305	40	2000
UTAS92	622	21266	1314	24249	38	2800
CARS91	682	16926	1385	29814	51	1550

Table 9.6. Carter's real-life exam timetable problems.

in the 'GA Avg' (average over the five runs) and 'GA Best' columns show the numbers of near-clashes occurring. In every case there were no violations of hard constraints and no violations of room capacities. It is also worth noting that some published results by other methods show instead the average weighted number of near-clashes per student in which, for example having adjacent exams attracts a weight of 16, having a gap of one free period attracts a weight of 8 and so on. These results are therefore very promising, even though it may in some cases be possible to reduce the number of slots used by ignoring the minimise-near-clashes preference. The 'Best Strategy' column shows the strategy used, eg 'BR(7,1)-BT(0,1) WL-24' means that in phase 1 a Brelaz-like strategy was used with heuristic 7 for exam-choosing and heuristic 1 for slot-choosing; in phase 2 a backtracking strategy was used employing heuristic 0 for exam-choosing and heuristic 1 for slot-choosing; phase 1 ended after placing the top 24% of large exams.

In every case the GA was limited to a maximum of 625 evaluations; the approach is fast.

It is interesting to observe the variation in best strategies, and in particular that in five cases there was effectively only a single phase.

6. Discussion and future directions

The above examples have shown that combining simple heuristics in a suitable sequence can be a good method of tackling complex problems. EAs prove to be a useful tool for searching for such sequences. So far,

Problem	Slots	Seats	GA Avg.	GA Best	Best Strategy
HECS92	21	1250	190	154	BR(7,1)-BT(0,1) WL-24
STAF83	15	600	932	821	BR(8,2)-BT(3,0) Wα-127
YORF83	21	500	764	708	BR(0,2)-FC(2,1) Wα-119
UTES92	12	1250	632	594	BR(2,0)-BT(1,1) Wα-16
EARF83	24	700	723	723	FC(4,0)
TRES92	27	655	599	586	FC(4,1)-BT(3,0) WL-25
LSEF91	21	900	247	221	BR(8,0)
KFUS93	24	1955	231	223	BR(1,0)-FC(3,0) Wα-97
RYES93	27	2500	754	671	BR(8,1)
CARF92	40	2000	285	285	BR(2,0)
UTAS92	38	2800	936	902	BR(0,0)-BR(6,2) Wα-262
CARS91	51	1550	170	130	BR(8,0)

Table 9.7. Results of evolving heuristic choices for timetabling

some limited experimenting by us suggests that combining heuristics often beats using a single heuristic throughout. This is tested by seeding the initial population with chromosomes that represent single-heuristic solutions and observing that evolution kills them off.

However, the specific idea of using a chromosome that encodes the choice of heuristic for each choice-point seems imperfect. We suspect that it works as well as it does because in many cases several heuristics would each have produced the same choice at that decision-point. Some investigation of the matter with the HGA for job-shop scheduling tends to confirm this. That does suggest that the approach might break down if the set of heuristics used was appreciably larger and more diverse; an encoding that lists the heuristic to use at each choice-point would then represent a colossal search space, perhaps without much duplication.

There are some other drawbacks to the EA approach which we intend to address in future work. Although the hyper-heuristic approach is an improvement on the traditional 'black-box' EA, in that the solution gives some information as to how the solution was created in terms of the heuristics used, it still does not associate those heuristics with the conditions that caused the heuristics to be invoked. Furthermore, an EA approach which encodes the choice of heuristic at every decision point is somewhat clumsy due to the risk of *epistasis*; each decision potentially affects all subsequent decisions hence changing one decision can have a catastrophic effect on the subsequent meaning of the chromosome. A more useful hyper-heuristic approach would be one which produced rules

of the form 'under condition X apply heuristic Y'. An obvious candidate for discovering rules of this form is to use classifier systems, and we intend to investigate this approach in the future. Another possibility is to investigate the use of genetic-programming techniques to evolve ranking functions for each heuristic, that give an indication of when they should be applied.

Finally, it should be intuitively clear that combining heuristics, at least in the ways illustrated in this chapter, is unlikely to produce optimal answers to problems. For many problems the cost of finding optima can be very high, and it is usually necessary to build a lot of problem- or domain-specific knowledge into the algorithms that aim to find optima. A key point of hyper-heuristic methods is instead to improve on known heuristics by combining them so that they make up for each others weaknesses. EAs provide a good framework for doing that.

Acknowledgments

The authors gratefully acknowledge EPSRC support through research grant GR/N36660.

References

Ausiello, G., Crescenzi, P., Gambosi, G., Kahn, V., Marchetti-Spaccamela, A., and Protasi, M. (1999). *Complexity and Approximation: Combinatorial Optimisation Problems And Their Approximability Properties.* Springer-Verlag, Berlin.

Brelaz, D. (1979). New methods to colour the vertices of a graph. *Communications of the ACM,* 22.

Bruns, R. (1993). Direct chromosome representation and advanced genetic algorithms for production scheduling. In Forrest, S., editor, *Proceedings of the Fifth International Conference on Genetic Algorithms,* San Mateo: Morgan Kaufmann, 352.

Bruns, R. (1997). Scheduling. In Bäck, T., Fogel, D., and Michalewicz, Z., editors, *Handbook of Evolutionary Computation,* release 97/1, chapter Part F: Applications of Evolutionary Computing, pages Fl.5:1–9. IOP Publishing Ltd and Oxford University Press.

Burke, E. and Ross, P. (1996). *Practice and Theory of Automated Timetabling.* Lecture Notes in Computer Science 1153. Springer, Berlin. First International Conference, Aug-Sep 1995.

Carter, M. (1998). *Practice and Theory of Automated Timetabling.* Lecture Notes in Computer Science. Springer, Berlin. Second International Conference, Aug 1997.

Carter, M. W., Laporte, C., and Lee, S. Y. (1995). Examination timetabling: Algorithmic strategies and applications. Working Paper 94-03, University of Toronto Dept of Industrial Engineering.

Coffman, E., Garey, M., and D.S.Johnson (1997). Approximation algorithms for bin-packing - a survey. In Hochbaum, D., editor, *Approximation Algorithms for NP-hard Problems,* PWS, Boston, 46–93.

Culberson, J. C. (1992). Iterated greedy graph coloring and the difficulty landscape. Technical Report TR 92-07, Edmonton, Alberta Canada T6G 2H1. ftp ftp.cs.ualberta.ca pub/TechReports.

Dorndorf, U. and Pesch, E. (1995). Evolution based learning in a job shop scheduling environment. *Computers and Operations Research,* 22(1):25–40.

Hart, E. and Ross, P. (1998). A heuristic combination method for solving jobshop scheduling problems. In A.E.Eiben, T.Back, M.Schoenauer, and H-P.schwefel, editors, *Parallel Problem Solving From Nature - PPSN V.* Springer-Verlag.

Hart, E. and Ross, P. (2000). A systematic investigation of ga performance on job shop scheduling problems. In et. al., S. C., editor, *Real-World Applications of Evolutionary Computing.* Springer-Verlag.

Fang, H.-L. (1994). *Genetic Algorithms in Timetabling and Scheduling.* PhD thesis, Department of Artificial Intelligence, University of Edinburgh.

Fang, H.-L., Ross, P., and Corne, D. (1993). A promising genetic algorithm approach to job-shop scheduling, rescheduling, and open-shop scheduling problems. In Forrest, S., editor, *Proceedings of the Fifth International Conference on Genetic Algorithms,* San Mateo: Morgan Kaufmann, 375–382.

Hoffman, P. (1998). *The Man Who Loved Only Numbers.* Fourth Estate, London.

Johnson, D. (1972). Near-optimal bin packing algorithms. Technical Report TR-109, MIT Computing Laboratory, Cambridge, Mass.

Katz, J. and McCormick, D. (1997). Genetic algorithms and rule-based systems. *Technical Analysis of Stocks and Commodities,* pages 46–60.

Katz, J. and McCormick. D. (2000). *The Encyclopaedia of Trading Strategies.* Irwin Trader's Edge. McGraw Hill, New York.

Lin, S.-C., Goodman, E., and Punch, W. (1997). A genetic algorithm approach to dynamic job-shop scheduling problems. In Back, T., editor, *Proceedings of the Seventh International Conference on Genetic Algorithms,* Morgan-Kaufmann, 481–489.

Minton, S., Johnston, M. D., Phillips, A. B., and Laird, P. (1992). Minimizing conflicts: a heuristic repair method for constraint satisifaction and scheduling problems. *Artificial Intelligence,* 58:161–205.

M.Vasquez and D.Whitley (2000). A comparison of genetic algorithms for the dynamic job-shop scheudling problem. In et. al., D., editor, *Proceedings of the Genetic and Evolutionary Computation Conference GECCO 2000.* Morgan-Kaufmann.

Nakano, R. and Yamada, T. (1991). Conventional genetic algorithms for job shop problems. In Belew, R. and Booker, L., editors, *Proceedings of the Fourth International Conference on Genetic Algorithms,* San Mateo: Morgan Kaufmann, 474–479.

Norenkov, I. and Goodman, E. (1997). Solving scheduling problems via evolutionary methods for rule sequence optimization. Second World Conference on Soft Computing.

Petford, A. and Welsh, D. (1989). A randomised 3-colouring algorithm. *Discrete Mathematics,* 74:253–261.

Ross, P., Corne, D., and Hart, E. (1997). Some observations about GA-based exam timetabling. In *Proceedings of the Second Conference on the Practice and Theory of A utomated Timetabling,* Toronto, Canada.

Schaffer, J. and Eshelman, L. (1996). Combinatorial optimisation by genetic algorithms: the value of the genotype/phenotype distinction. In Goodman, E., Uskov, V., and Punch III, W., editors, *EvCA96: Proceedings of the First International Conference on Evolutionary Computation and its Applications,* Moscow. Institute for High-performance Computer Systems, Russian Academy of Sciences.

Terashima-Marin, H., Ross, P., and Valenzuela-Rendon, M. (1999). Evolution of constraint-satisfaction strategies in examination timetabling. In Banzhaf, W., Daida, J., Eiben, A., Garzon, M., Honavar, V., Jakiela, M., and Smith, R., editors, *Proceedings of the Genetic and Evolutionary Computation Conference: GECCO-99,* San Mateo, CA. Morgan Kaufmann,635–642.

Vaessens, R., Aarts, E., and Lenstra, J. (1996). Job shop scheduling by local search. *INFORMS Journal of Computing,* 8:302–317.

Chapter 10

CONSTRAINED GENETIC ALGORITHMS AND THEIR APPLICATIONS IN NONLINEAR CONSTRAINED OPTIMIZATION *

Benjamin W. Wah and
Yi-Xin Chen

Abstract This chapter presents a framework that unifies various search mechanisms for solving constrained nonlinear programming (NLP) problems. These problems are characterized by functions that are not necessarily differentiable and continuous. Our proposed framework is based on the first-order necessary and sufficient condition developed for constrained local minimization in discrete space that shows the equivalence between discrete-neighborhood saddle points and constrained local minima. To look for discrete-neighborhood saddle points, we formulate a discrete constrained NLP in an augmented Lagrangian function and study various mechanisms for performing ascents of the augmented function in the original-variable subspace and descents in the Lagrange-multiplier subspace. Our results show that *CSAGA,* a combined constrained simulated annealing and genetic algorithm, performs well when using crossovers, mutations, and annealing to generate trial points. Finally, we apply iterative deepening to determine the optimal number of generations in *CSAGA* and show that its performance is robust with respect to changes in population size.

1. Introduction

Many engineering applications can be formulated as constrained *nonlinear programming problems* (NLPs). Examples include production planning, computer integrated manufacturing, chemical control process-

*Research supported by National Aeronautics and Space Administration Contract NAS2-37143.

ing, and structure optimization. These applications can be solved by existing methods if they are specified in well-defined formulae that are differentiate and continuous. However, only special cases can be solved when they do not satisfy the required assumptions. For instance, sequential quadratic programming cannot handle problems whose objective and constraint functions are not differentiate or whose variables are discrete or mixed. Since many applications involving optimization may be formulated by non-differentiable functions with discrete or mixed-integer variables, it is important to develop new methods for handling these optimization problems, inxxconstrained nonlinear programming

The study of algorithms for solving a disparity of constrained optimization problems is difficult unless the problems can be represented in a unified way. In this chapter we assume that continuous variables are first discretized into discrete variables in such a way that the values of functions using discretized variables approach those of the original continuous variables. Such an assumption is valid when continuous variables are represented as floating-point numbers and when the range of variables is small (say between 10^{-5} and 10^{5}). Intuitively, if discretization is fine enough, then solutions found in discretized space are fairly good approximations to the original solutions. The accuracy of solutions found in discretized problems has been studied elsewhere (Wu, 2000).

Based on discretization, continuous and mixed-integer constrained NLPs can be represented as discrete constrained NLPs as follows:[1]

$$\begin{array}{lll} \text{minimize} & f(x) & \\ \text{subject to} & g(x) \leq 0 & x = [x_1, \ldots, x_n]^T \text{ is a vector} \\ & h(x) = 0 & \text{of bounded discrete variables.} \end{array} \tag{10.1}$$

Here, $f(x)$ is a lower-bounded objective function, $g(x) = [g_1(x), \ldots, g_k(x)]^T$ is a vector of k inequality constraints, $h(x) = [h_1(x), \cdots, h_m(x)]^T$ is a vector of m equality constraints. Functions $f(x)$, $g(x)$, and $h(x)$ are not necessarily differentiable and can be either linear or nonlinear, continuous or discrete, and analytic or procedural. Without loss of generality, we consider only minimization problems.

Solutions to (10.1) cannot be characterized in ways similar to those of problems with differentiable functions and continuous variables. In the latter class of problems, solutions are defined with respect to neighborhoods of open spheres with radius approaching zero asymptotically. Such a concept does not exist in problems with discrete variables.

[1] For two vectors v and w of the same number of elements, $v \leq w$ means that each element of v is not less than the corresponding element of w. $v \geq w$ can be defined similarly. 0, when compared to a vector, stands for a null vector.

Let X be the Cartesian product of the discrete domains of all variables in x. To characterize solutions sought in discrete space, we define the following concepts on neighborhoods and constrained solutions in discrete space:

Definition 1. $\mathcal{N}_{dn}(x)$, the *discrete neighborhood* (Aarts and Korst, 1989) of point $x \in X$ is a *finite* user-defined set of points $\{x' \in X\}$ such that $x' \in \mathcal{N}_{dn}(x) \Longleftrightarrow x \in \mathcal{N}_{dn}(x')$, and that for any $y^1, y^k \in X$, it is possible to find a finite sequence of points in X, $y^1, \cdots, y^k$, such that $y^{i+1} \in \mathcal{N}_{dn}(y^i)$ for $i = 1, \cdots k-1$.

Definition 2. Point $x \in X$ is called a *constrained local minimum in discrete neighborhood* (CLM_{dn}) if it satisfies two conditions: a) x is feasible, and b) $f(x) \leq f(x')$, for all feasible $x' \in \mathcal{N}_{dn}(x)$.

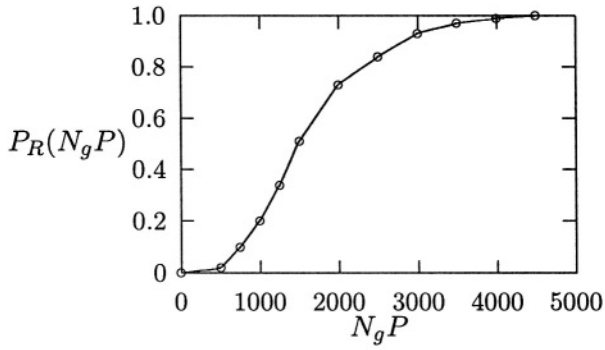
Definition 3. Point $x \in X$ is called a *constrained global minimum in discrete neighborhood* (CGM_{dn}) iff a) x is feasible, and b) for every feasible point $x' \in X$, $f(x') \geq f(x)$. The set of all CGM_{dn} is X_{opt}. According to our definitions, a CGM_{dn} must also be a CLM_{dn}.

In a similar way, there are definitions on continuous-neighborhood constrained local minima (CLM_{cn}) and constrained global minima (CGM_{cn}).

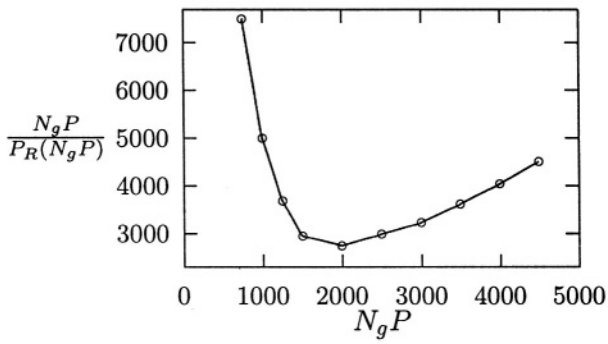
We have shown earlier (Wah and Wu, 1999) that the necessary and sufficient condition for a point to be a CLM_{dn} is that it satisfies the discrete-neighborhood saddle-point condition (Section 2.1). We have also extended simulated annealing (*SA*) (Wah and Wang, 1999) and greedy search (Wah and Wu, 1999) to look for discrete-neighborhood saddle points SP_{dn} (Section 2.2). At the same time, new problem-dependent constraint-handling heuristics have been developed in the *GA* community to handle nonlinear constraints (Michalewicz and Schoenauer, 1996) (Section 2.3). Up to now, there is no clear understanding on how to unify these algorithms into one that can be applied to find CGM_{dn} for a wide range of problems.

Based on our previous work, our goal in this chapter is to develop an effective framework that unifies *SA, GA,* and greedy search for finding CGM_{dn}. In particular, we propose *constrained genetic algorithm* (*CGA*) and *combined constrained SA and GA* (*CSAGA*) that look for SP_{dn}. We also study algorithms with the optimal average completion time for finding a CGM_{dn}.

The algorithms studied in this chapter are all stochastic searches that probe a search space in a random order, where a probe is a neighboring point examined by an algorithm, independent of whether it is accepted or not. Assuming p_j to be the probability that an algorithm finds a CGM_{dn} in its j^{th} probe and a simplistic assumption that all probes are



a) $P_R(N_gP)$ approaches one asymptotically



b) Existence of absolute minimum $N_{opt}P$ in $\frac{N_gP}{P_R(N_gP)}$

Figure 10.1. An example showing the application of *CSAGA* with $P = 3$ to solve a discretized version of G1 (Michalewicz and Schoenauer, 1996) ($N_{opt}P \approx 2000$).

independent, the performance of one run of such an algorithm can be characterized by N, the number of probes made (or CPU time taken), and $P_R(N)$, the *reachability probability* that a CGM_{dn} is hit in any of the N probes:

$$P_R(N) = 1 - \prod_{j=1}^{N}(1 - p_j), \quad \text{where } N \geq 0. \tag{10.2}$$

Reachability can be maintained by reporting the best solution found by the algorithm when it stops.

As an example, Figure 10.1a plots $P_R(N_gP)$ when *CSAGA* (see Section 3.2) was run under various number of generations N_g and fixed population size $P = 3$ (where $N = N_gP$). The graph shows that $P_R(N_gP)$ approaches one asymptotically as N_gP is increased.

Although it is hard to estimate the value of $P_R(N)$ when a test problem is solved by an algorithm, we can always improve the chance of finding a solution by running the same algorithm multiple times, each with N probes, from random starting points. Given $P_R(N)$ for one run of the algorithm and that all runs are independent, the expected total number of probes to find a CGM_{dn} is:

$$\sum_{j=1}^{\infty} P_R(N)(1 - P_R(N))^{j-1} N \times j = \frac{N}{P_R(N)}. \tag{10.3}$$

Figure 10.1b plots (10.3) based on $P_R(N_g P)$ in Figure 10.1a. In general, there exists N_{opt} that minimizes (10.3) because $P_R(0) = 0$, $\lim_{N_\alpha \to \infty} P_R(N_\alpha) = 1$, $\frac{N}{P_R(N)}$ is bounded below by zero, and $\frac{N}{P_R(N)} \to \infty$ as $N \to \infty$. The curve in Figure 10.1b illustrates this behavior.

Based on the existence of N_{opt}, we present in Section 3.3 search strategies in *CGA* and in *CSAGA* that minimize (10.3) in finding a CGM_{dn}. Finally, Section 4 compares the performance of our algorithms.

2. Previous Work

In this section, we first summarize the theory of Lagrange multipliers applied to solve (10.1) and two algorithms developed based on the theory. We then describe existing work in GA for solving constrained NLPs.

2.1 Theory of Lagrange multipliers for solving discrete constrained NLPs

Define a discrete equality-constrained NLP as follows (Wah and Wu, 1999; Wu, 2000):

$$\begin{array}{lll} \min\limits_{x} & f(x) & x \text{ is a vector of bounded} \\ \text{subject to} & h(x) = 0 & \text{discrete variables,} \end{array} \tag{10.4}$$

A *generalized discrete augmented Lagrangian function* of (10.4) is defined as follows (Wah and Wu, 1999):

$$L_d(x, \lambda) = f(x) + \lambda^T H(h(x)) + \frac{1}{2}||h(x)||^2, \tag{10.5}$$

where H is a non-negative continuous transformation function satisfying $H(y) \geq 0$, $H(y) = 0$ iff $y = 0$, and $\lambda = [\lambda_1, \cdots, \lambda_m]^T$ is a vector of Lagrange multipliers.

Function H is easy to design; examples include $H(h(x)) = [|h_1(x)|, \cdots, |h_m(x)|]^T$ and $H(h(x)) = [\max(h_1(x), 0), \cdots, \max(h_m(x), 0)]^T$. Note

that these transformations are not used in Lagrange-multiplier methods in continuous space because they are not differentiable at $H(h(x)) = 0$. However, they do not pose problems here because we do not require their differentiability.

Similar transformations can be used to transform inequality constraint $g_j(x) \leq 0$ into equivalent equality constraint $\max(g_j(x), 0) = 0$. Hence, we only consider problems with equality constraints from here on.

We define a *discrete-neighborhood saddle point* $SP_{dn}(x^*, \lambda^*)$ with the following property:

$$L_d(x^*, \lambda) \leq L_d(x^*, \lambda^*) \leq L_d(x, \lambda^*) \tag{10.6}$$

for all $x \in \mathcal{N}_{dn}(x^*)$ and all $\lambda, \lambda' \in R^m$. Note that although we use similar terminologies as in continuous space, SP_{dn} is different from SP_{cn} (saddle point in continuous space) because they are defined using different neighborhoods.

The concept of SP_{dn} is very important in discrete problems because, starting from them, we can derive first-order necessary and sufficient condition for CLM_{dn} that leads to global minimization procedures. This is stated formally in the following theorem (Wah and Wu, 1999):

Theorem 1 First-order necessary and sufficient condition on CLM_{dn} *(Wah and Wu, 1999). A point in the discrete search space of (10.4) is a* CLM_{dn} *iff it satisfies (10.6) for any* $\lambda \geq \lambda^*$

Theorem 1 is stronger than its continuous counterparts. The first-order necessary conditions in continuous Lagrange-multiplier theory (Luenberger, 1984) require CLM_{cn} to be regular points and functions to be differentiable. In contrast, there are no such requirements for CLM_{dn}. Further, the first-order conditions in continuous theory (Luenberger, 1984) are only necessary, and second-order sufficient condition must be checked in order to ensure that a point is actually a CLM_{cn} (*CLM* in continuous space). In contrast, the condition in Theorem 1 is necessary as well as sufficient.

2.2 Existing algorithms for finding SP_{dn}

Since there is a one-to-one correspondence between CGM_{dn} and SP_{dn}, it implies that a strategy looking for SP_{dn} with the minimum objective value will result in CGM_{dn}. We review two methods to look for SP_{dn}.

The first algorithm is the *discrete Lagrangian method* (DLM) (Wu, 1998). It is an iterative local search that looks for SP_{dn} by updating the variables in x in order to perform descents of L_d in the x subspace, while occasionally updating the λ variables of unsatisfied constraints in

1. **procedure CSA** (α, N_α)
2. set initial $\mathbf{x} \leftarrow [x_1, \cdots, x_n, \lambda_1, \cdots, \lambda_k]^T$ with random x, $\lambda \leftarrow 0$;
3. **while** stopping condition is not satisfied **do**
4. generate $\mathbf{x}' \in \mathcal{N}_{dn}(\mathbf{x})$ using $G(\mathbf{x}, \mathbf{x}')$;
5. accept $\mathbf{x}'$ with probability $A_T(\mathbf{x}, \mathbf{x}')$
6. reduce temperature by $T \leftarrow \alpha T$;
7. **end_while**
8. **end_procedure**

a) *CSA* called with schedule N_α and rate α

1. **procedure** CSA_{ID}
2. set initial cooling rate $\alpha \leftarrow \alpha_0$ and $N_\alpha \leftarrow N_{\alpha_0}$;
3. set $K \leftarrow$ number of CSA runs at fixed α;
4. **repeat**
5. **for** $i \leftarrow 1$ **to** K **do call** $CSA(\alpha, N_\alpha)$; **end_for**;
6. increase cooling schedule $N_\alpha \leftarrow \rho N_\alpha$;
7. **until** feasible solution has been found **and** no better solution in two successive increases of N_α;
8. **end_procedure**

b) CSA_{ID}: *CSA* with iterative deepening

Figure 10.2. Constrained simulated annealing algorithm (CSA) and its iterative-deepening extension

order to perform ascents in the λ subspace and to force the violated constraints into satisfaction. When no new probes can be generated in both the x and λ subspaces, the algorithm has additional mechanisms to escape from such local traps. It can be shown that the point where *DLM* stops is a CLM_{dn} when the number of neighborhood points is small enough to be enumerated in each descent in the x subspace (Wah and Wu, 1999; Wu, 2000). However, when the number of neighborhood points is very large and hill-climbing is used to find the first point with an improved L_d in each descent, then the point where *DLM* stops may be a feasible point but not necessarily a SP_{dn}.

The second algorithm is the *constrained simulated annealing* (CSA) (Wah and Wang, 1999) algorithm shown in Figure 10.2a. It looks for SP_{dn} by probabilistic descents in the x subspace and by probabilistic ascents in the λ subspace, with an acceptance probability governed by the Metropolis probability. Similar to DLM, if the neighborhood of every point is very large and cannot be enumerated, then the point where *CSA* stops may only be a feasible point but not necessarily a SP_{dn}.

Using $G(\mathbf{x}, \mathbf{x}')$ for generating trial point $\mathbf{x}'$ in $\mathcal{N}_{dn}(\mathbf{x})$, $A_T(\mathbf{x}, \mathbf{x}')$ as the Metropolis acceptance probability, and a logarithmic cooling schedule,

CSA has been proven to have asymptotic convergence with probability one to CGM_{dn} (Wah and Wang, 1999). This property is stated in the following theorem:

Theorem 2 *Asymptotic convergence of CSA.* The Markov chain modeling *CSA* converges to a CGM_{dn} with probability one.

Theorem 2 extends a similar theorem for *SA* that proves its asymptotic convergence to unconstrained global minima of unconstrained optimization problems. By looking for SP_{dn} in the Lagrangian-function space, Theorem 2 shows the asymptotic convergence of CSA to CGM_{dn} in constrained optimization problems.

Theorem 2 implies that *CSA* is not a practical algorithm when used to find CGM_{dn} in one run with certainty because *CSA* will take infinite time.

In practice, when *CSA* is run once using a a finite cooling schedule N_α, it finds a CGM_{dn} with reachability probability $P_R(N_\alpha) < 1$. To increase its success probability, *CSA* with a finite N_α can be run multiple times from random starting points. Assuming that all the runs are independent, a CGM_{dn} can be found in finite average time defined by (10.3).

We have verified experimentally that the expected time defined in (10.3) has an absolute minimum at N_{opt}. (Figure 10.1b illustrates the existence of N_{opt} for *CSAGA*.) It follows that, in order to minimize (10.3), *CSA* should be run multiple times from random starting points using schedule N_{opt}.

To find N_{opt} at run time without using problem-dependent information, we have proposed to use *iterative deepening* (Korf, 1985) by starting *CSA* with a short schedule and by doubling the schedule each time the current run fails to find a CGM_{dn} (Wah and Chen, 2000). Since the total overhead in iterative deepening is dominated by that of the last run, CSA_{ID} (CSA with iterative deepening in Figure 10.2b) has a completion time of the same order of magnitude as that using N_{opt} when the last schedule that *CSA* is run is close to N_{opt} and that this run succeeds. Figure 10.3 illustrates that the total time incurred by CSA_{ID} is of the same order as the expected overhead at N_{opt}.

Note that $P_R(N_{opt}) < 1$ for one run of *CSA* at N_{opt}. When *CSA* is run with a schedule close to N_{opt} and fails to find a solution, its cooling schedule will be doubled and overshoots beyond N_{opt}. To reduce the chance of overshooting into exceedingly long cooling schedules and to increase the success probability before its schedule reaches N_{opt}, we have proposed to run *CSA* multiple times from random starting points at each schedule in CSA_{ID}. Figure 10.2b shows *CSA* that is run $K = 3$

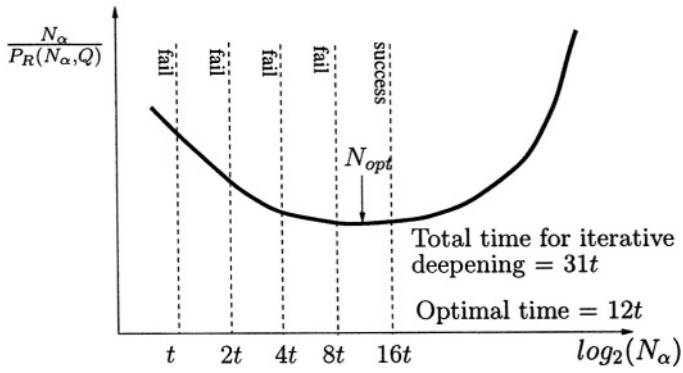


Figure 10.3. An application of iterative deepening in CSA_{ID}.

times at each schedule before the schedule is doubled. Our results show that such a strategy generally requires twice the average completion time with respect to multiple runs of CSA using N_{opt} before it finds a CGM_{dn} (Wah and Chen, 2000).

2.3 Genetic algorithms for solving constrained NLP problems

Genetic algorithm (GA) is a general stochastic optimization algorithm that maintains a population of alternative candidates and that probes a search space using genetic operators, such as crossovers and mutations, in order to find better candidates. The original GA was developed for solving unconstrained problems, using a single fitness function to rank candidates. Recently, many variants of GA have been developed for solving constrained NLPs. Most of these methods were based on penalty formulations that use GA to minimize an unconstrained penalty function $\mathcal{F}(x)$, consisting of a sum of the objective and the constraints weighted by penalties. Similar to CSA, these methods do not require the differentiability or continuity of functions.

One penalty formulation is the *static-penalty formulation* in which all penalties are fixed (Bertsekas, 1982):

$$\mathcal{F}_\rho(x,\gamma) = f(x) + \sum_{i=1}^{m} \gamma_i |h_i(x)|^\rho, \tag{10.7}$$

where $\rho > 0$, and penalty vector $\gamma = \{\gamma_1, \gamma_2, \cdots, \gamma_m\}$ is *fixed* and chosen to be large enough so that

$$\mathcal{F}_\rho(x^*,\gamma) < \mathcal{F}_\rho(x,\gamma) \quad \forall x \in X - X_{opt} \text{ and } x^* \in X_{opt}. \tag{10.8}$$

Based on (10.8), an unconstrained global minimum of (10.7) over x is a CGM_{dn} to (10.4); hence, it suffices to minimize (10.7) in solving (10.4). Since both $f(x)$ and $|h_i(x)|$ are lower bounded and x takes finite discrete values, γ always exists and is finite, thereby ensuring the correctness of the approach. Note that other forms of penalty formulations have also been studied in the literature.

The major issue of static-penalty methods lies in the difficulty of selecting a suitable γ. If γ is much larger than necessary, then the terrain will be too rugged to be searched effectively by local-search methods. If it is too small, then feasible solutions to (10.7) may be difficult to find.

Dynamic-penalty methods (Joines and Houck, 1994), on the other hand, address the difficulties of static-penalty methods by increasing penalties gradually in the following fitness function:

$$\mathcal{F}(x) = f(x) + (C \times t)^{\alpha} \sum_{j=1}^{m} |h_i(x)|^{\beta}, \tag{10.9}$$

where t is the generation number, and C, α, and β are constants. In contrast to static-penalty methods, $(C \times t)^{\alpha}$, the penalty on infeasible points, is increased during evolution.

Dynamic-penalty methods do not always guarantee convergence to CLM_{dn} or CGM_{dn}. For example, consider a problem with two constraints $h_1(x) = 0$ and $h_2(x) = 0$. Assuming that a search is stuck at an infeasible point x' and that for all $x \in \mathcal{N}_{dn}(x')$, $0 < |h_1(x')| < |h_1(x)|$, $|h_2(x')| > |h_2(x)| > 0$, and $|h_1(x')|^{\beta} + |h_2(x')|^{\beta} < |h_1(x)|^{\beta} + |h_2(x)|^{\beta}$, then the search can never escape from x' no matter how large $(C \times t)^{\alpha}$ grows.

One way to ensure the convergence of dynamic-penalty methods is to use a different penalty for each constraint, as in Lagrangian formulation (10.5). In the previous example, the search can escape from x' after assigning a much larger penalty to $h_2(x')$ than that to $h_1(x')$.

There are many other variants of penalty methods, such as annealing penalties, adaptive penalties (Michalewicz and Schoenauer, 1996) and self-adapting weights (Eiben and Ruttkay, 1996). In addition, problem-dependent operators have been studied in the GA community for handling constraints. These include methods based on preserving feasibility with specialized genetic operators, methods searching along boundaries of feasible regions, methods based on decoders, repair of infeasible solutions, co-evolutionary methods, and strategic oscillation. However, most methods require domain-specific knowledge or problem-dependent genetic operators, and have difficulties in finding feasible regions or in maintaining feasibility for nonlinear constraints.

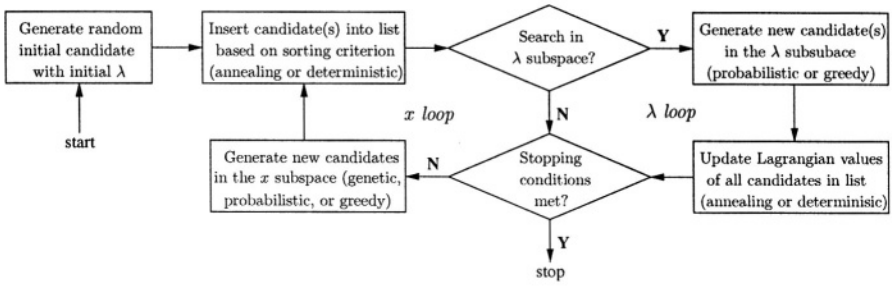


Figure 10.4. An iterative stochastic procedural framework to look for SP_{dn}.

In general, local minima of penalty functions are only necessary but not sufficient to be constrained local minima of the original constrained optimization problems, unless the penalties are chosen properly. Hence, finding local minima of a penalty function does not necessarily solve the original constrained optimization problem.

3. A General Framework to look for SP_{dn}

Although there are many methods for solving constrained NLPs, our survey in the last section shows a lack of a general framework that unifies these mechanisms. Without such a framework, it is difficult to know whether different algorithms are actually variations of each other. In this section we present a framework for solving constrained NLPs that unifies *SA, GA,* and greedy searches.

Based on the necessary and sufficient condition in Theorem 1, Figure 10.4 depicts a stochastic procedure to look for SP_{dn}. The procedure consists of two loops: the x loop that updates the variables in x in order to perform descents of L_d in the x subspace, and the λ loop that updates the λ variables of unsatisfied constraints for any candidate in the list in order to perform ascents in the λ subspace. The procedure quits when no new probes can be generated in both the x and λ subspaces.

The procedure will not stop until it finds a feasible point because it will generate new probes in the λ subspace when there are unsatisfied constraints. Further, if the procedure always finds a descent direction at x by enumerating all points in $\mathcal{N}_{dn}(x)$, then the point where the procedure stops must be a feasible local minimum in the x subspace of $L_d(x, \lambda)$, or equivalently, a CLM_{dn}.

Both *DLM* and *CSA* discussed in Section 2.2 fit into this framework, each maintaining a list of one candidate at any time. DLM entails greedy searches in the x and λ subspaces, deterministic insertions into

```
1. procedure CGA(P, N_g)
2.     set generation number t ← 0 and λ(t) ← 0;
3.     initialize population P(t);
4.     repeat /* over multiple generations */
5.         evaluate L_d(x, λ(t)) for all candidates in P(t);
6.         repeat /* over probes in x subspace */
7.             y ← GA(select(P(t)));
8.             evaluate L_d(y, λ) and insert into P(t)
9.         until sufficient probes in x subspace;
10.        λ(t) ← λ(t) ⊕ cH(h, P(t)); /* update λ */
11.        t ← t + 1;
12.    until (t > N_g)
13. end_procedure
```

a) *CGA* called with population size P and number of generations N_g.

```
1. procedure CGA_ID
2.     set initial number of generations N_g = N_0;
3.     set K = number of CGA runs at fixed N_g;
3.     repeat /* iterative deepening to find CGM_dn */
4.         for i ← 1 to K do call CGA(P, N_g) end_for
5.         set N_g ← ρN_g (typically ρ = 2);
6.     until N_g exceeds maximum allowed or
           (no better solution has been found in two
           successive increases of N_g and N_g > ρ^5 N_0
           and a feasible solution has been found);
7. end_procedure
```

b) CGA_{ID}: *CGA* with iterative deepening

Figure 10.5. Constrained *GA* and its iterative deepening version.

the list of candidates, and deterministic acceptance of candidates until all constraints are satisfied. On the other hand, *CSA* generates new probes randomly in one of the x or λ variables, accepts them based on the Metropolis probability if L_d increases along the x dimension and decreases along the λ dimension, and stops updating λ when all constraints are satisfied.

In this section, we use genetic operators to generate probes and present in Section 3.1 *CGA* and in Section 3.2 *CSAGA*. Finally, we propose in Section 3.3 iterative-deepening versions of these algorithms.

3.1 Constrained genetic algorithm (CGA)

CGA in Figure 10.5a was developed based on the general framework in Figure 10.4. Similar to traditional *GA*, it organizes a search into a number of generations, each involving a population of candidate points

in a search space. However, it searches in the L_d space using genetic operators to generate probes in the x subspace, either greedy or probabilistic mechanisms to generate probes in the λ subspace, and deterministic organization of candidates according to their L_d values.

Lines 2-3 initialize to zero the generation number t and the vector of Lagrange multipliers λ. The initial population $\mathcal{P}(t)$ can be either randomly generated or user provided.

Lines 4 and 12 terminate *CGA* when the maximum number of allowed generations is exceeded.

Line 5 evaluates in generation t all candidates in $\mathcal{P}(t)$ using $L_d(x, \lambda(t))$ as the fitness function.

Lines 6-9 explore the x subspace by selecting from $\mathcal{P}(t)$ candidates to reproduce using genetic operators and by inserting the new candidates generated into $\mathcal{P}(t)$ according to their fitness values.

After a number of descents in the x subspace (defined by the number of probes in Line 9 and the decision box "search in λ subspace?" in Figure 10.4), the algorithm switches to searching in the λ subspace. Line 10 updates λ according to the vector of maximum violations $\mathcal{H}(h(x), \mathcal{P}(t))$, where the maximum violation of a constraint is evaluated over all the candidates in $\mathcal{P}(t)$. That is,

$$\mathcal{H}_i(h(x), \mathcal{P}(t)) = \max_{x \in \mathcal{P}(t)} H(h_i(x)),\ i = 1, \cdots, m, \qquad (10.10)$$

where $h_i(x)$ is the i^{th} constraint function, H is the non-negative transformation defined in (10.5), and c is a step-wise constant controlling how fast λ changes.

Operator $\oplus$ in Figure 10.5a can be implemented in two ways in order to generate a new λ. A new λ can be generated probabilistically based a uniform distribution in $[\frac{-c\mathcal{H}}{2}, \frac{c\mathcal{H}}{2}]$, or in a greedy fashion based on a uniform distribution in $(0, c\mathcal{H}]$. In addition, we can accept new probes deterministically by rejecting negative ones, or probabilistically using an annealing rule. In all cases, a Lagrange multiplier will not be changed if its corresponding constraint is satisfied.

3.2 Combined Constrained *SA* and *GA* (*CSAGA*)

Based on the general framework in Figure 10.4, we design *CSAGA* by integrating *CSA* in Figure 10.2a and *CGA* in Figure 10.5a into a combined procedure. The new procedure differs from the original *CSA* in two respects. First, by maintaining multiple candidates in a population, we need to decide how *CSA* should be applied to the multiple candidates in a population. Our evaluations show that, instead of running

```
1.  procedure CSAGA(P, N_g)
2.      set t ← 0, T_0, 0 < α < 1, and P(t);
3.      repeat /* over multiple generations */
4.          for i ← 1 to q do /* SA in Lines 5-10 */
5.              for j ← 1 to P do
6.                  generate x'_j from N_dn(x_j) using G(x_j, x'_j);
7.                  accept x'_j with probability A_T(x_j, x'_j)
8.              end_for
9.              set T ← αT; /* set T for the SA part */
10.         end_for
11.         repeat /* by GA over probes in x subspace */
12.             y ← GA(select(P(t)));
13.             evaluate L_d(y, λ) and insert y into P(t);
14.         until sufficient number of probes in x subspace;
15.         t ← t + q; /* update generation number */
16.     until (t ≥ N_g)
17. end_procedure
```

Figure 10.6. CSAGA: Combined *CSA* and *CGA* called with population size P and N_g generations.

CSA corresponding to a candidate from a random starting point, it is best to run *CSA* sequentially, using the best solution point found in one run as the starting point of the next run. Second, we need to determine the duration of each run of *CSA*. This is controlled by parameter q that was set to be $\frac{N_g}{6}$ after experimental evaluations. The new algorithm shown in Figure 10.6 uses both *SA* and *GA* to generate new probes in the x subspace.

Line 2 initializes $\mathcal{P}(0)$. Unlike *CGA*, any candidate $\mathbf{x} = [x_1, \cdots, x_n, \lambda_1, \cdots, \lambda_k]^T$ in $\mathcal{P}(t)$ is defined in the joint x and λ subspaces. Initially, x can be user-provided or randomly generated, and λ is set to zero.

Lines 4-10 perform *CSA* using q probes on every candidate in the population. In each probe, we generate probabilistically $\mathbf{x}'_j$ and accept it based on the Metropolis probability. Experimentally, we set q to be $\frac{N_g}{6}$. As discussed earlier, we use the best point of one run as the starting point of the next run.

Lines 11-15 start a *GA* search after the *SA* part has been completed. The algorithm searches in the x subspace by generating probes using *GA* operators, sorting all candidates according to their fitness values L_d after each probe is generated. In ordering candidates, since each candidate has its own vector of Lagrange multipliers, the algorithm first computes the average value of Lagrange multipliers for each constraint over all candidates in $\mathcal{P}(t)$ and then calculates L_d for each candidate using the average Lagrange multipliers.

Note that *CSAGA* has difficulties similar to those of *CGA* in determining a proper number of candidates to use in its population and the duration of each run. We address these two issues in the $CSAGA_{ID}$ in the next subsection.

3.3 *CGA* and *CSAGA* with iterative deepening

In this section we present a method to determine the optimal number of generations in one run of *CGA* and *CSAGA* in order to find a CGM_{dn}. The method is based on the use of iterative deepening (Korf, 1985) that determines an upper bound on N_g in order to minimize the expected total overhead in (10.3), where N_g is the number of generations in one run of *CGA*.

The number of probes expended in one run of *CGA* or *CSAGA* is $N = N_g P$, where P is the population size. For a fixed P, let $\hat{P}_R(N_g) = P_R(PN_g)$ be the reachability probability of finding CGM_{dn}. From (10.3), the expected total number of probes using multiple runs of either *CGA* or *CSAGA* and fixed P is:

$$\frac{N}{P_R(N)} = \frac{N_g P}{P_R(N_g P)} = P\frac{N_g}{\hat{P}_R(N_g)} \tag{10.11}$$

In order to have an optimal number of generations $N_{g_{opt}}$ that minimizes (10.11), $\frac{N_g}{\hat{P}_R(N_g)}$ must have an absolute minimum in $(0,\infty)$. This condition is true since $\hat{P}_R(N_g)$ of *CGA* has similar behavior as $P_R(N_g)$ of *CSA*. It has been verified based on statistics collected on $\hat{P}_R(N_g)$ and N_g at various P when *CGA* and *CSAGA* are used to solve ten discretized benchmark problems G1-G10 (Michalewicz and Schoenauer, 1996). Figure 10.1b illustrates the existence of such an absolute minimum when *CSAGA* with $P = 3$ was applied to solve G1.

Similar to the design of CSA_{ID}, we apply iterative deepening to estimate $N_{g_{opt}}$. CGA_{ID} in Figure 10.5b uses a set of geometrically increasing N_g to find a CGM_{dn}:

$$N_{g_i} = \rho^i N_0, \qquad i = 0, 1, \ldots \tag{10.12}$$

where N_0 is the (small) initial number of generations.

Under each N_g, *CGA* is run for a maximum of K times but stops immediately when a feasible solution has been found, when no better solution has been found in two successive generations, and after the number of iterations has been increased geometrically at least five times. These conditions are used to ensure that iterative deepening has been applied adequately. For iterative deepening to work, $\rho > 1$.

Let $\hat{P}_R(N_{g_i})$ be the reachability probability of one run of *CGA* under N_{g_i} generations, $B_{opt}(f')$ be the expected total number of probes taken by *CGA* with $N_{g_{opt}}$ to find a CGM_{dn}, and $\mathcal{B}_{ID}(f')$ be the expected total number of probes taken by CGA_{ID} in Figure 10.5b to find a solution of quality f' starting from N_0 generations. According to (10.11),

$$B_{opt}(f') = P\frac{N_{g_{opt}}}{\hat{P}_R(N_{g_{opt}})} \tag{10.13}$$

The following theorem shows the sufficient conditions in order for $\mathcal{B}_{ID}(f') = O(B_{opt}(f'))$.

Theorem 3 Optimality of CGA_{ID} and $CSAGA_{ID}$.
$\mathcal{B}_{ID}(f') = O(B_{opt}(f'))$ *if*

a) $\hat{P}_R(0) = 0$; $\hat{P}_R(N_g)$ *is monotonically non-decreasing for* N_g *in* $(0, \infty)$; *and* $\lim_{N_g \to \infty} \hat{P}_R(N_g) \leq 1$;

b) $(1 - \hat{P}_R(N_{g_{opt}}))^K \rho < 1$.

The proof is not shown due to space limitations.

Typically, $\rho = 2$, and $\hat{P}_R(N_{g_{opt}}) \geq 0.25$ in all the benchmarks tested. Substituting these values into condition (b) in Theorem 3 yields $K > 2.4$. In our experiments, we have used $K = 3$. Since *CGA* is run a maximum of three times under each N_g, $B_{opt}(f')$ is of the same order of magnitude as *one* run of *CGA* with $N_{g_{opt}}$.

The only remaining issue left in the design of CGA_{ID} and $CSAGA_{ID}$ is in choosing a suitable population size P in each generation.

In designing CGA_{ID}, we found that the optimal P ranges from 4 to 40 and is difficult to determine a priori. Although it is possible to choose a suitable P dynamically, we do not present the algorithm here due to space limitations and because it performs worse than $CSAGA_{ID}$.

In selecting P for $CSAGA_{ID}$, we note in the design of CSA_{ID} in Figure 10.2b that $K = 3$ parallel runs are made at each cooling schedule in order to increase the success probability of finding a solution. For this reason, we set $P = K = 3$ in our experiments. Our experimental results in the next section show that, although the optimal P may be slightly different from 3, the corresponding expected overhead to find a CGM_{dn} differs very little from that when a constant P *is* used.

4. Experimental Results

We present our experimental results in evaluating CSA_{ID}, CGA_{ID} and $CSAGA_{ID}$ on discrete constrained NLPs. Based on the framework

in Figure 10.4, we first determine the best combination of strategies to use in generating probes and in organizing candidates. Using the best combination of strategies, we then show experimental results on some constrained NLPs.

Due to a lack of large-scale discrete benchmarks, we derive our benchmarks from two sets of continuous benchmarks: Problem G1-G10 (Michalewicz and Schoenauer, 1996; Koziel and Michalewicz, 1999) and Floudas and Pardalos' Problems (Floudas and Pardalos, 1990).

4.1 Implementation Details

In theory, algorithms derived from the framework, such as *C S A, CGA,* and *CSAGA,* will look for SP_{dn}. In practice, however, it is important to choose appropriate neighborhoods and generate proper trial points in x and λ subspaces in order to solve constrained NLPs efficiently.

An important component of these methods is the frequency at which λ is updated. Like in *CSA* (Wah and Wang, 1999), we have set experimentally in *CGA* and *CSAGA* the ratio of generating trial points in x and λ subspaces from the current point to be 20n to m, where n is the number of variables and m is the number of constraints. This ratio means that x is updated more often than λ.

In generating trial points in the x subspace, we have used a dynamically controlled neighborhood size in the SA part (Wah and Wang, 1999) based on the 1:1 ratio rule (Corana et al., 1987), whereas in the GA part, we have used the seven operators in Genocop III (Michalewicz and Nazhiyath, 1995) and L_d as our fitness function. In implementing CSA_{ID}, CGA_{ID} and $CSAGA_{ID}$, we have used the default parameters of CSA (Wah and Wang, 1999) in the SA part and those of Genocop III (Michalewicz and Nazhiyath, 1995) in the GA part.

The generation of trial point λ' in the λ subspace is done by the following rule:

$$\lambda'_j = \lambda_j + r_1 \, \phi_j \qquad \text{where } j = 1, \cdots, m. \tag{10.14}$$

Here, r_1 is randomly generated in [–1/2, +1/2] if we choose to generate λ probabilistically, and is randomly generated in [0, 1] if we choose to generate probes in λ in a greedy fashion.

We adjust ϕ adaptively according to the degree of constraint violations, where

$$\phi = w \otimes \mathcal{H}(x) = [w_1 \mathcal{H}_1(x), w_2 \mathcal{H}_2(x), \cdots, w_m \mathcal{H}_m(x)], \tag{10.15}$$

$\otimes$ represents vector product, and $\mathcal{H}$ is the vector of maximum violations defined in (10.10). When $\mathcal{H}_i(x)$ is satisfied, λ_i does not need to be

Table 10.1. Timing results on evaluating various combinations of strategies in CSA_{ID}, CGA_{ID} and $CSAGA_{ID}$ with $P = 3$ to find solutions that deviate by 1% and 10% from the best-known solution of a discretized version of G2. All CPU times in seconds were averaged over 10 runs and were collected on a Pentinum III 500-MHz computer with Solaris 7. '–' means that no solution with desired quality can be found.

Probe Generation Strategy		Insertion	Sol. 1% off CGM_{dn}			Sol. 10% off CGM_{dn}		
λ subspace	x subspace	Strategy	*CSA*	*CGA*	*CSAGA*	*CSA*	*CGA*	*CSAGA*
probabilistic	probabilistic	annealing	6.91	23.99	4.89	1.35	–	1.03
probabilistic	probabilistic	deterministic	9.02	–	6.93	1.35	2.78	1.03
probabilistic	deterministic	annealing	–	18.76	–	89.21	2.40	–
probabilistic	deterministic	deterministic	–	16.73	–	–	2.18	–
greedy	probabilistic	annealing	7.02	–	7.75	1.36	–	0.90
greedy	probabilistic	deterministic	7.02	–	7.75	1.36	–	0.90
greedy	deterministic	annealing	–	25.50	–	82.24	1.90	–
greedy	deterministic	deterministic	–	25.50	–	82.24	1.90	–

updated; hence, $\phi_i = 0$. In contrast, when a constraint is not satisfied, we adjust ϕ_i by modifying w_i according to how fast $\mathcal{H}_i(x)$ is changing:

$$w_i = \begin{cases} \eta_0 \, w_i & \text{if } \mathcal{H}_i(x) > \tau_0 T \\ \eta_1 \, w_i & \text{if } \mathcal{H}_i(x) < \tau_1 T \end{cases} \tag{10.16}$$

where T is the temperature, and $\eta_0 = 1.25$, η_1=0.8, $\tau_0 = 1.0$, and $\tau_1 = 0.01$ were chosen experimentally. When $\mathcal{H}_i(x)$ is reduced too quickly (*i.e.*, $\mathcal{H}_i(x) < \tau_1 T$), $\mathcal{H}_i(x)$ is over-weighted, leading to possibly poor objective values or difficulty in satisfying other under-weighted constraints. Hence, we reduce λ_i's neighborhood. In contrast, if $\mathcal{H}_i(x)$ is reduced too slowly (*i.e.*, $\mathcal{H}_i(x) > \tau_0 T$), we enlarge λ_i's neighborhood in order to improve its chance of satisfaction. Note that w_i is adjusted using T as a reference because constraint violations are expected to decrease when T decreases.

In addition, for iterative deepening to work, we have set the following parameters: $\rho = 2$, $K = 3$, $N_0 = 10 \cdot n_v$, and $N_{max} = 1.0 \times 10^8 n_v$, where n_v is the number of variables, and N_0 and N_{max} are, respectively, initial and maximum number of probes.

4.2 Evaluation Results

Due to a lack of large-scale discrete benchmarks, we derive our benchmarks from two sets of continuous benchmarks: Problem G1-G10 (Michalewicz and Schoenauer, 1996; Koziel and Michalewicz, 1999) and Floudas and Pardalos' Problems (Floudas and Pardalos, 1990).

In generating a discrete constrained NLP, we discretize continuous variables in the original continuous constrained NLP into discrete vari-

ables as follows. In discretizing continuous variable x_i in range $[l_i, u_i]$, where l_i and u_i are lower and upper bounds of x_i, respectively, we force x_i to take values from the set:

$$A_i = \begin{cases} \left\{a_i + \frac{b_i - a_i}{s} j,\ j = 0, 1, \cdots, s\right\} & \text{if } b_i - a_i < 1 \\ \left\{a_i + \frac{1}{s} j,\ j = 0, 1, \cdots, \lfloor (b_i - a_i)s \rfloor\right\} & \text{if } b_i - a_i \geq 1, \end{cases} \tag{10.17}$$

where $s = 1.0 \times 10^7$.

Table 10.1 shows the results of evaluating various combinations of strategies in CSA_{ID}, CGA_{ID}, and $CSAGA_{ID}$ on a discretized version of G2 (Michalewicz and Schoenauer, 1996; Koziel and Michalewicz, 1999). We show the average time of 10 runs for each combination in order to reach two solution quality levels (1% or 10% worse than CGM_{dn}, assuming the value of CGM_{dn} is known). Evaluation results on other benchmark problems are similar and are not shown due to space limitations.

Our results show that CGA_{ID} is usually less efficient than CSA_{ID} or $CSAGA_{ID}$. Further, CSA_{ID} or $CSAGA_{ID}$ has better performance when probes generated in the x subspace are accepted by annealing rather than by deterministic rules (the former prevents a search from getting stuck in local minima or infeasible points). On the other hand, there is little difference in performance when new probes generated in the λ subspace are accepted by probabilistic or by greedy rules and when new candidates are inserted according to annealing or deterministic rules. In short, generating probes in the x and λ subspaces probabilistically and inserting candidates in both the x and λ subspaces by annealing rules leads to good and stable performance. For this reason, we use this combination of strategies in our experiments.

We next test our algorithms on ten constrained NLPs G1-G10 (Michalewicz and Schoenauer, 1996; Koziel and Michalewicz, 1999). These problems have objective functions of various types (linear, quadratic, cubic, polynomial, and nonlinear) and constraints of linear inequalities, nonlinear equalities, and nonlinear inequalities. The number of variables is up to 20, and that of constraints, including simple bounds, is up to 42. The ratio of feasible space with respect to the whole search space varies from 0% to almost 100%, and the topologies of feasible regions are quite different. These problems were originally designed to be solved by evolutionary algorithms (EAs) in which constraint handling techniques were tuned for each problem in order to get good results. Examples of such techniques include keeping a search within feasible regions with specific genetic operators and dynamic and adaptive penalty methods.

Table 10.2. Results on CSA_{ID}, CGA_{ID} and $CSAGA_{ID}$ in finding the best-known solution f^* for 10 discretized constrained NLPs and their corresponding results found by EA. (S.T. stands for strategic oscillation, H.M. for homomorphous mappings, and D.P. for dynamic penalty. $\mathcal{B}_{ID}(f^*)$, the CPU time in seconds to find the best-known solution f^*, were averaged over 10 runs and were collected on a Pentinum III 500-MHz computer with Solaris 7. The best $\mathcal{B}_{ID}(f^*)$ for each problem is boxed.)

Prob.	Best	EAs		CSA_{ID}	CGA_{ID}		$CSAGA_{ID}$			
ID	Sol. f^*	Best f	Method	$\mathcal{B}_{ID}(f^*)$	P_{opt}	$\mathcal{B}_{ID}(f^*)$	P	$\mathcal{B}_{ID}(f^*)$	P_{opt}	$\mathcal{B}_{ID}(f^*)$
G1	-15	-15	Genocop	1.65	40	5.49	3	1.64	2	1.31
G2	-0.80362	0.803553	S.T.	7.28	30	311.98	3	5.18	3	5.18
G3	1.0	1.0	S.T.	1.07	30	14.17	3	0.89	3	0.89
G4	-30665.5	-30664.5	H.M.	0.76	5	3.95	3	0.95	3	0.95
G5	4221.9	5126.498	D.P.	2.88	30	68.9	3	2.76	2	2.08
G6	-6961.81	-6961.81	Genocop	0.99	4	7.62	3	0.91	2	0.73
G7	24.3062	24.62	H.M.	6.51	30	31.60	3	4.60	4	4.07
G8	0.095825	0.095825	H.M.	0.11	30	0.31	3	0.13	4	0.10
G9	680.63	680.64	Genocop	0.74	30	5.67	3	0.57	3	0.57
G10	7049.33	7147.9	H.M.	3.29	30	82.32	3	3.36	3	3.36

Table 10.2 compares the performance of CSA_{ID}, CGA_{ID}, and $CSAGA_{ID}$ with respect to $\mathcal{B}_{ID}(f^*)$, the expected total CPU time of multiple runs until a solution of value f^* is found. The first two columns show the problem IDs and the corresponding known f^*. The next two columns show the best solutions obtained by EAs and the specific constraint handling techniques used to generate the solutions. Since all CSA_{ID}, CGA_{ID} and $CSAGA_{ID}$ can find a CGM_{dn} in *all 10 runs,* we compare their performance with respect to T, the average total overhead of multiple runs until a CGM_{dn} is found. The fifth and sixth columns show, respectively, the average time and number of $L_d(x, \lambda)$ function evaluations CSA_{ID} takes to find f^*. The next two columns show the performance of CGA_{ID} with respect to P_{opt}, the optimal population size found by enumeration, and the average time to find f^*. These results show that CGA_{ID} is not competitive as compared to CSA_{ID}, even when P_{opt} is used. The results on including additional steps in CGA_{ID} to select a suitable P at run time are worse and are not shown due to space limitations. Finally, the last five columns show the performance of $CSAGA_{ID}$. The first three present the average times and number of $L_d(x, \lambda)$ evaluations using a constant P, whereas the last two show the average times using P_{opt} found by enumeration. These results show little improvements in using P_{opt}. Further, $CSAGA_{ID}$ has between 9% and 38% in improvement in $\mathcal{B}_{ID}(f^*)$, when compared to that of CSA_{ID}, for the 10 problems except for G4 and G10.

Table 10.3. Results on CSA_{ID} and $CSAGA_{ID}$ with $P = 3$ in solving selected Floudas and Pardalos' discretized constrained NLP benchmarks (with more than $n_v = 10$ variables). Since Problem 5.* and 7.* are especially large and difficult and a search can rarely reach their true CGM_{dn}, we consider a CGM_{dn} found when the solution quality is within 10% of the true CGM_{dn}. All CPU times in seconds were averaged over 10 runs and were collected on a Pentium-III 500-MHz computer with Solaris 7.

Problem		$f(x)$	CSA_{ID}	$CSAGA_{ID}$
ID	Best f^*	n_v	$\mathcal{B}_{ID}(f^*)$	$\mathcal{B}_{ID}(f^*)$
2.7.1(min)	-394.75	20	35.11 sec.	14.86 sec.
2.7.2(min)	-884.75	20	53.92 sec.	15.54 sec.
2.7.3(min)	-8695.0	20	34.22 sec.	22.52 sec.
2.7.4(min)	-754.75	20	36.70 sec.	16.20 sec.
2.7.5(min)	-4150.4	20	89.15 sec.	23.46 sec.
5.2(min)	1.567	46	3168.29 sec.	408.69 sec.
5.4(min)	1.86	32	2629.52 sec.	100.66 sec.
7.2(min)	1.0	16	824.45 sec.	368.72 sec.
7.3(min)	1.0	27	2323.44 sec.	1785.14 sec.
7.4(min)	1.0	38	951.33 sec.	487.13 sec.

Comparing CGA_{ID} and $CSAGA_{ID}$ with EA, we see that EA was only able to find f^* in three of the ten problems, despite extensive tuning and using problem-specific heuristics, whereas both CGA_{ID} and $CSAGA_{ID}$ can find f^* for all these problems without any problem-dependent strategies. It is not possible to report the timing results of EA because the results are the best among many runs after extensive tuning.

Finally, Table 10.3 shows the results on selected discretized Floudas and Pardalos' benchmarks (Floudas and Pardalos, 1990) that have more than 10 variables and that have many equality or inequality constraints. The first three columns show the problem IDs, the known f^*, and the number of variables (n_v) in each. The last two columns compare $\mathcal{B}_{ID}(f^*)$ of CSA_{ID} and $CSAGA_{ID}$ with fixed $P = 3$. They show that $CSAGA_{ID}$ is consistently faster than CSA_{ID} (between 1.3 and 26.3 times), especially for large problems. This is attributed to the fact that *GA* maintains more diversity of candidates by keeping a population, thereby allowing competition among the candidates and leading *SA* to explore more promising regions.

5. Conclusions

In this chapter we have presented new algorithms to look for discrete-neighborhood saddle points in discrete Lagrangian space of constrained

optimization problems. Our results show that genetic algorithms, when combined with simulated annealing, are effective in locating saddle points. Future developments will focus on better ways to select appropriate heuristics in probe generation, including search direction control and neighborhood size control, at run time.

References

Aarts, E. and Korst, J. (1989) *Simulated Annealing and Boltzmann Machines.* J. Wiley and Sons.

Bertsekas, D. P. (1982) *Constrained Optimization and Lagrange Multiplier Methods.* Academic Press.

Corana, A., Marchesi, M., Martini, C. and Ridella, S. (1987) Minimizing multimodal functions of continuous variables with the simulated annealing algorithm. *ACM Trans. on Mathematical Software,* 13(3):262–280.

Eiben, A. E. and Ruttkay, Zs. (1996) Self-adaptivity for constraint satisfaction: Learning penalty functions. *Proceedings of the 3rd IEEE Conference on Evolutionary Computation,* 258–261.

Floudas, C. A. and Pardalos, P. M. (1990) *A Collection of Test Problems for Constrained Global Optimization Algorithms,* volume 455 of *Lecture Notes in Computer Science.* Springer-Verlag.

Joines, J. and Houck, C. (1994) On the use of non-stationary penalty functions to solve nonlinear constrained optimization problems with gas. *Proceedings of the First IEEE International Conference on Evolutionary Computation,* 579–584.

Korf, R. E. (1985) Depth-first iterative deepening: An optimal admissible tree search. *Artificial Intelligence,* 27:97–109.

Koziel, S. and Michalewicz, Z. (1999) Evolutionary algorithms, homomorphous mappings, and constrained parameter optimization. *Evolutionary Computation,* 7(l):19–44.

Luenberger, D. G. (1984) *Linear and Nonlinear Programming.* Addison-Wesley Publishing Company, Reading, MA.

Michalewicz, Z. and Nazhiyath, G. (1995) Genocop III: A co-evolutionary algorithm for numerical optimization problems with nonlinear constraints. *Proceedings of IEEE International Conference on Evolutionary Computation,* 2:647–651.

Michalewicz, Z. and Schoenauer, M. (1996) Evolutionary algorithms for constrained parameter optimization problems. *Evolutionary Computation,* 4(l):l–32.

Wah, B. W. and Chen, Y. X. (2000) Optimal anytime constrained simulated annealing for constrained global optimization. *Sixth Int'l Conf. on Principles and Practice of Constraint Programming.*

Wah, B. W. and Wang, T. (1999) Simulated annealing with asymptotic convergence for nonlinear constrained global optimization. *Principles and Practice of Constraint Programming,* 461–475.

Wah, B. W. and Wu, Z. (1999) The theory of discrete Lagrange multipliers for nonlinear discrete optimization. *Principles and Practice of Constraint Programming,* 28–42.

Wu, Z. (1998) *Discrete Lagrangian Methods for Solving Nonlinear Discrete Constrained Optimization Problems.* M.Sc. Thesis, Dept. of Computer Science, Univ. of Illinois, Urbana, IL.

Wu, Z. (2000) *The Theory and Applications of Nonlinear Constrained Optimization using Lagrange Multipliers.* Ph.D. Thesis, Dept. of Computer Science, Univ. of Illinois, Urbana, IL.

This page intentionally left blank

V

PARAMETER SELECTION IN EAS

This page intentionally left blank

Chapter 11

PARAMETER SELECTION

Zbigniew Michalewicz,
Ágoston E. Eiben and
Robert Hinterding

Abstract The issue of parameter selection in an evolutionary algorithm is one of the most important elements in building a successful application. In this chapter we revise the terminology connected with parameter selection and control and discuss a classification of various methods which have been studied by the evolutionary computation community in recent years.

1. Introduction

The two major steps in applying any heuristic search algorithm to a particular problem are the specification of the representation and the evaluation (fitness) function. These two items form the bridge between the original problem context and the problem-solving framework. When defining an evolutionary algorithm (EA) one needs to choose its components, such as variation operators (mutation and recombination) that suit the representation, selection mechanisms for selecting parents and survivors, and an initial population. Each of these components may have parameters, for instance: the probability of mutation, the tournament size of selection, or the population size. The values of these parameters greatly determine whether the algorithm will find a near-optimum solution, and whether it will find such a solution efficiently. Choosing the right parameter values, however, is a time-consuming task and considerable effort has gone into developing good heuristics for it.

Globally, we distinguish two major forms of setting parameter values: parameter *tuning* and parameter *control.* By parameter tuning we mean the commonly practised approach that amounts to finding good values for the parameters *before* the run of the algorithm and then running the algorithm using these values, which remain fixed during the run. In

Section 2 we give arguments that any static set of parameters, having the values fixed during an EA run, seems to be inappropriate. Parameter control forms an alternative, as it amounts to starting a run with initial parameter values which are changed *during* the run.

In this chapter (which is based on our earlier article (Eiben et al., 1999)) we provide a comprehensive discussion of parameter control and categorize different ways of performing it. The classification is based on two aspects: *how* the mechanism of change works, and *what* component of the EA is effected by the mechanism. Such a classification can be useful to the evolutionary computation community, since many researchers interpret terms like "adaptation" or "self-adaptation" differently, which can be confusing. The framework proposed in (Eiben et al., 1999) was intended to eliminate ambiguities in the terminology.

Some other classification schemes were proposed, e.g., (Angeline, 1995; Hinterding et al., 1997; Smith and Fogarty, 1997), that use other division criteria, resulting in different classification schemes. The classification of Angeline (1995) is based on levels of adaptation and type of update rules. In particular, three levels of adaptation: population-level, individual-level, and component-level[1] are considered, together with two types of update mechanisms: absolute and empirical rules. Absolute rules are predetermined and specify how modifications should be made. On the other hand, empirical update rules modify parameter values by competition among them (self-adaptation). Angeline's framework considers an EA as a whole, without dividing attention to its different components (e.g., mutation, recombination, selection, etc). The classification proposed by Hinterding, Michalewicz, and Eiben (1997) extends that of (Angeline, 1995) by considering an additional level of adaptation (environment-level), and makes a more detailed division of types of update mechanisms, dividing them into deterministic, adaptive, and self-adaptive categories. Here again, no attention is payed to what parts of an EA are adapted. The classification of Smith and Fogarty (Smith and Fogarty, 1997; Smith, 1997) is probably the most comprehensive. It is based on three division criteria: what is being adapted, the scope of the adaptation, and the basis for change. The last criterion is further divided into two categories: the evidence the change is based upon and the rule/algorithm that executes the change. Moreover, there are two types of rule/algorithm: uncoupled/absolute and tightly-coupled/empirical, the latter one coinciding with self-adaptation.

[1]Notice, that we use the term 'component' differently from (Angeline, 1995) where Angeline denotes subindividual structures with it, while we refer to parts of an EA, such as operators (mutation, recombination), selection, fitness function, etc.

The classification scheme discussed here is based on the type of update mechanisms and the EA component that is adapted, as basic division criteria. This classification addresses the key issues of parameter control without getting lost in details (this aspect is discussed in more detail in section 4).

This chapter is organized as follows. The next section discusses parameter tuning and parameter control. Section 3 presents an example which provides some basic intuitions on parameter control. Section 4 develops a classification of control techniques in evolutionary algorithms, whereas Section 5 discusses briefly various techniques.

2. Parameter tuning vs. parameter control

During the 1980s, a standard genetic algorithm (GA) based on bit-representation, one-point crossover, bit-flip mutation and roulette wheel selection (with or without elitism) was widely applied. Algorithm design was thus limited to choosing the so-called control parameters, or strategy parameters[2], such as mutation rate, crossover rate, and population size. Many researchers based their choices on tuning the control parameters "by hand", that is experimenting with different values and selecting the ones that gave the best results. Later, they reported their results of applying a particular EA to a particular problem, paraphrasing here:

> ...for these experiments, we have used the following parameters: population size of 100, probability of crossover equal to 0.85, etc.

without much justification of the choice made.

Two main approaches were tried to improve GA design in the past. First, De Jong (1975) put a considerable effort into finding parameter values (for a traditional GA), which were good for a number of numeric test problems. He determined (experimentally) recommended values for the probabilities of single-point crossover and bit mutation. His conclusions were that the following parameters give reasonable performance for his test functions (for new problems these values may not be very good):

> population size of 50
> probability of crossover equal to 0.6
> probability of mutation equal to 0.001
> generation gap of 100%
> scaling window: $n = \infty$
> selection strategy: elitist.

Grefenstette (1986), on the other hand, used a GA as a meta-algorithm to optimize values for the same parameters for both on-line and off-line

[2]By 'control parameters' or 'strategy parameters' we mean the parameters of the EA, not those of the problem.

performance[3] of the algorithm. The best set of parameters to optimize the on-line (off-line) performance of the GA were (the values to optimize the off-line performance are given in parenthesis):

population size of 30 (80)
probability of crossover equal to 0.95 (0.45)
probability of mutation equal to 0.01 (0.01)
generation gap of 100% (90%)
scaling window: $n = 1$ ($n = 1$)
selection strategy: elitist (non-elitist).

Note that in both of these approaches, an attempt was made to find the optimal and *general* set of parameters; in this context, the word 'general' means that the recommended values can be applied to a wide range of optimization problems. Formerly, genetic algorithms were seen as robust problem solvers that exhibit approximately the same performance over a wide range of problems (Goldberg, 1989), pp. 6. The contemporary view on EAs, however, acknowledges that specific problems (problem types) require specific EA setups for satisfactory performance (Bäck et al., 1997). Thus, the scope of 'optimal' parameter settings is necessarily narrow. Any quest for generally (near-)optimal parameter settings is lost *a priori* (Wolpert and Macready, 1997). This stresses the need for efficient techniques that help finding good parameter settings for a given problem, in other words, the need for good parameter tuning methods.

As an alternative to tuning parameters before running the algorithm, controlling them during a run was realised quite early (e.g., mutation step sizes in the evolution strategy (ES) community). Analysis of the simple corridor and sphere problems in large dimensions led to Rechenberg's 1/5 success rule (see section 3.1), where feedback was used to control the mutation step size (Rechenberg, 1973). Later, self-adaptation of mutation was used, where the mutation step size and the preferred direction of mutation were controlled without any direct feedback. For certain types of problems, self-adaptive mutation was very successful and its use spread to other branches of evolutionary computation (EC).

As mentioned earlier, parameter tuning by hand is a common practice in evolutionary computation. Typically one parameter is tuned at a time, which may cause some sub-optimal choices, since parameters often interact in a complex way. Simultaneous tuning of more parameters, however, leads to an enormous amount of experiments. The technical

[3]These measures were defined originally by De Jong (De Jong, 1975); the intuition is that on-line performance is based on monitoring the best solution in each generation, while off-line performance takes all solutions in the population into account.

drawbacks to parameter tuning based on experimentation can be summarized as follows:

- Parameters are not independent, but trying all different combinations systematically is practically impossible.
- The process of parameter tuning is time consuming, even if parameters are optimized one by one, regardless to their interactions.
- For a given problem the selected parameter values are not necessarily optimal, even if the effort made for setting them was significant.

Other options for designing a good set of static parameters for an evolutionary method to solve a particular problem include "parameter setting by analogy" and the use of theoretical analysis. Parameter setting by analogy amounts to the use of parameter settings that have been proved successful for "similar" problems. However, it is not clear whether similarity between problems as perceived by the user implies that the optimal set of EA parameters is also similar. As for the theoretical approach, the complexities of evolutionary processes and characteristics of interesting problems allow theoretical analysis only after significant simplifications in either the algorithm or the problem model. Therefore, the practical value of the current theoretical results on parameter settings is unclear. There are some theoretical investigations on the optimal population size (Goldberg, 1989; Thierens, 1996; Harik et al., 1997; Goldberg et al., 1992) or optimal operator probabilities (Goldberg et al., 1991; Theirens and Goldberg, 1991; Bäck, 1993; Schaffer and Morishima, 1987), however, these results were based on simple function optimization problems and their applicability for other types of problems is limited.

A general drawback of the parameter tuning approach, regardless of how the parameters are tuned, is based on the observation that a run of an EA is an intrinsically dynamic, adaptive process. The use of rigid parameters that do not change their values is thus in contrast to this spirit. Additionally, it is intuitively obvious that different values of parameters might be optimal at different stages of the evolutionary process (Davis, 1989; Syswerda, 1991; Bäck, 1992; Bäck, 1992; Bäck, 1993; Hesser and Männer, 1991; Soule and Foster, 1997). For instance, large mutation steps can be good in the early generations helping the exploration of the search space and small mutation steps might be needed in the late generations to help fine tuning the sub-optimal chromosomes. This implies that the use of static parameters itself can lead to inferior algorithm performance. The straightforward way to treat this problem is by using parameters that may change over time, that is, by replacing a parame-

ter p by a function $p(t)$, where t is the generation counter. However, as indicated earlier, the problem of finding optimal *static* parameters for a particular problem can be quite difficult, and the optimal values may depend on many other factors (like the applied recombination operator, the selection mechanism, etc). Hence designing an optimal function $p(t)$ may be even more difficult. Another possible drawback to this approach is that the parameter value $p(t)$ changes are caused by a deterministic rule triggered by the progress of time t, without taking any notion of the actual progress in solving the problem, i.e., without taking into account the current state of the search. Yet researchers have improved their evolutionary algorithms, i.e., they improved the quality of results returned by their algorithms while working on particular problems, by using such simple deterministic rules. This can be explained simply by superiority of changing parameter values: suboptimal choice of $p(t)$ often leads to better results than a suboptimal choice of p.

To this end, recall that finding good parameter values for an evolutionary algorithm is a poorly structured, ill-defined, complex problem. But on this kind of problem, EAs are often considered to perform better than other methods! It is thus seemingly natural to use an evolutionary algorithm not only for finding solutions to a problem, but also for tuning the (same) algorithm to the particular problem. Technically speaking, this amounts to modifying the values of parameters during the run of the algorithm by taking the actual search process into account. Basically, there are two ways to do this. Either one can use some heuristic rule which takes feedback from the current state of the search and modifies the parameter values accordingly, or incorporate parameters into the chromosomes, thereby making them subject to evolution. The first option, using a heuristic feedback mechanism, allows one to base changes on triggers different from elapsing time, such as population diversity measures, relative improvements, absolute solution quality, etc. The second option, incorporating parameters into the chromosomes, leaves changes entirely based on the evolution mechanism. In particular, natural selection acting on solutions (chromosomes) will drive changes in parameter values associated with these solutions. In the following we discuss these options illustrated by an example.

3. An example

Let us assume we deal with a numerical optimization problem:

$$\text{optimize } f(\mathbf{x}) = f(x_1, \ldots, x_n),$$

subject to some inequality and equality constraints:

$$g_i(\mathbf{x}) \leq 0 \ (i = 1, \ldots, q) \text{ and } h_j(\mathbf{x}) = 0 \ (j = q+1, \ldots, m),$$

and bounds $l_i \leq x_i \leq u_i$ for $1 \leq i \leq n$, defining the domain of each variable.

For such a numerical optimization problem we may consider an evolutionary algorithm based on a floating-point representation, where each individual $\mathbf{x}$ in the population is represented as a vector of floating-point numbers

$$\mathbf{x} = \langle x_1, \ldots, x_n \rangle.$$

3.1 Changing the mutation step size

Let us assume that we use Gaussian mutation together with arithmetical crossover to produce offspring for the next generation. A Gaussian mutation operator requires two parameters: the mean, which is often set to zero, and the standard deviation σ, which can be interpreted as the mutation step size. Mutations then are realized by replacing components of the vector $\mathbf{x}$ by

$$x_i' = x_i + N(0, \sigma),$$

where $N(0, \sigma)$ is a random Gaussian number with mean zero and standard deviation σ. The simplest method to specify the mutation mechanism is to use the same σ for all vectors in the population, for all variables of each vector, and for the whole evolutionary process, for instance, $x_i' = x_i + N(0, 1)$. As indicated in Section 2, it might be beneficial to vary the mutation step size.[4] We shall discuss several possibilities in turn.

First, we can replace the static parameter σ by a dynamic parameter, i.e., a function $\sigma(t)$. This function can be defined by some heuristic rule assigning different values depending on the number of generations. For example, the mutation step size may be defined as:

$$\sigma(t) = 1 - 0.9 \cdot \frac{t}{T},$$

where t is the current generation number varying from 0 to T, which is the maximum generation number. Here, the mutation step size $\sigma(t)$ (used for all vectors in the population and for all variables of each vector) will decrease slowly from 1 at the beginning of the run $(t = 0)$ to 0.1 as the number of generations t approaches T. Such decreases may assist the fine-tuning capabilities of the algorithm. In this approach, the value of the given parameter changes according to a fully deterministic scheme. The user thus has full control of the parameter and its value at a given time t is completely determined and predictable.

Second, it is possible to incorporate feedback from the search process, still using the same σ for all for vectors in the population and for all variables of each vector. A well-known example of this type of parameter

[4]There are even formal arguments supporting this view in specific cases, e.g., (Bäck, 1992; Bäck, 1992; Bäck, 1993; Hesser and Männer, 1991).

adaptation is Rechenberg's '1/5 success rule' in (1+1)-evolution strategies (Rechenberg, 1973). This rule states that the ratio of successful mutations[5] to all mutations should be 1/5, hence if the ratio is greater than 1/5 then the step size should be increased, and if the ratio is less than 1/5, the step size should be decreased:

$$\textbf{if } (t \mod n = 0) \textbf{ then}$$
$$\sigma(t) := \begin{cases} \sigma(t-n)/c, & \textbf{if } p_s > 1/5 \\ \sigma(t-n) \cdot c, & \textbf{if } p_s < 1/5 \\ \sigma(t-n), & \textbf{if } p_s = 1/5 \end{cases}$$
$$\textbf{else}$$
$$\sigma(t) := \sigma(t-1);$$
$$\textbf{fi}$$

where p_s is the relative frequency of successful mutations, measured over some number of generations and $0.817 \leq c \leq 1$, (Bäck, 1996). Using this mechanism, changes in the parameter values are now based on feedback from the search, and σ-adaptation happens every n generations. The influence of the user on the parameter values is much less direct here than in the deterministic scheme above. Of course, the mechanism that embodies the link between the search process and parameter values is still a heuristic rule indicating how the changes should be made, but the values of $\sigma(t)$ are not deterministic.

Third, it is possible to assign an 'individual' mutation step size to each solution: extend the representation to individuals of length $n+1$ as

$$\langle x_1, \ldots, x_n, \sigma \rangle,$$

and apply some variation operators (e.g., Gaussian mutation and arithmetical crossover) to x_i's as well as to the σ value of an individual. In this way, not only the solution vector values (x_i's), but also the mutation step size of an individual undergoes evolution. A typical variation would be:

$$\sigma' = \sigma \cdot e^{N(0,\tau_0)} \text{ and}$$
$$x_i' = x_i + N(0, \sigma'),$$

where τ_0 is a parameter of the method. This mechanism is commonly called self-adapting the mutation step sizes. Observe that within the self-adaptive scheme the heuristic character of the mechanism resetting the parameter values is eliminated.[6]

[5] A mutation is considered successful if it produces an offspring that is better than the parent.

[6] It can be argued that the heuristic character of the mechanism resetting the parameter values is not eliminated, but rather replaced by a metaheuristic of evolution itself. However, the method is very robust w.r.t. the setting of τ_0 and a good rule is $\tau_0 = 1/\sqrt{n}$.

Note that in the above scheme the scope of application of a certain value of σ was restricted to a single individual. However, it can be applied to all variables of the individual: it is possible to change the granularity of such applications and use a separate mutation step size to each x_i. If an individual is represented as

$$\langle x_1, \ldots, x_n, \sigma_1, \ldots, \sigma_n \rangle,$$

then mutations can be realized by replacing the above vector according to a similar formula as discussed above:

$$\sigma_i' = \sigma_i \cdot e^{N(0,\tau_0)}, \text{ and}$$
$$x_i' = x_i + N(0, \sigma_i'),$$

where τ_0 is a parameter of the method. However, as opposed to the previous case, each component x_i has its own mutation step size σ_i, which is being self-adapted. This mechanism implies a larger degree of freedom for adapting the search strategy to the topology of the fitness landscape.

3.2 Changing the penalty coefficients

In the previous subsection we described different ways to modify a parameter controlling mutation. Several other components of an EA have natural parameters, and these parameters are traditionally tuned in one or another way. Here we show that other components, such as the evaluation function (and consequently the fitness function) can also be parameterized and thus tuned. While this is a less common option than tuning mutation (although it is practicized in the evolution of variable-length stuctures for parsimony pressure (Zhang and Mühlenbein, 1995)), it may provide a useful mechanism for increasing the performance of an evolutionary algorithm.

When dealing with constrained optimization problems, penalty functions are often used. A common technique is the method of static penalties (Michalewicz and Schoenauer, 1996), which requires fixed user-supplied penalty parameters. The main reason for its wide spread use is that it is the simplest technique to implement: It requires only the straightforward modification of the evaluation function *eval* as follows:

$$eval(\mathbf{x}) = f(\mathbf{x}) + W \cdot penalty(\mathbf{x}),$$

where f is the objective function, and $penalty(\mathbf{x})$ is zero if no violation occurs, and is positive,[7] otherwise. Usually, the *penalty* function is based on the distance of a solution from the feasible region, or on the effort to "repair" the solution, i.e., to force it into the feasible region. In many methods a set of functions f_j $(1 \leq j \leq m)$ is used to construct the penalty, where the function f_j measures the violation of the j-th

[7] For minimization problems.

constraint in the following way:

$$f_j(\mathbf{x}) = \begin{cases} \max\{0, g_j(\mathbf{x})\}, & if\ 1 \le j \le q \\ |h_j(\mathbf{x})|, & if\ q+1 \le j \le m. \end{cases}$$

W is a user-defined weight, prescribing how severely constraint violations are weighted.[8] In the most traditional penalty approach the weight W does not change during the evolution process. We sketch three possible methods of changing the value of W.

First, we can replace the static parameter W by a dynamic parameter, e.g., a function $W(t)$. Just as for the mutation parameter σ, we can develop a heuristic which modifies the weight W over time. For example, in the method proposed by Joines and Houck (1994), the individuals are evaluated (at the iteration t) by a formula, where

$$eval(\mathbf{x}) = f(\mathbf{x}) + (C \cdot t)^{\alpha} \cdot penalty(\mathbf{x}),$$

where C and α are constants. Clearly,

$$W(t) = (C \cdot t)^{\alpha},$$

the penalty pressure grows with the evolution time.

Second, let us consider another option, which utilizes feedback from the search process. One example of such an approach was developed by Bean and Hadj-Alouane (1992), where each individual is evaluated by the same formula as before, but $W(t)$ is updated in every generation t in the following way:

$$W(t+1) = \begin{cases} (1/\beta_1) \cdot W(t), & \text{if } \mathbf{b}^i \in \mathcal{F} \\ & \text{for all } t-k+1 \le i \le t \\ \beta_2 \cdot W(t), & \text{if } \mathbf{b}^i \in \mathcal{S} - \mathcal{F} \\ & \text{for all } t-k+1 \le i \le t \\ W(t), & \text{otherwise.} \end{cases}$$

In this formula, $\mathcal{S}$ is the set of all search points (solutions), $\mathcal{F} \subseteq \mathcal{S}$ is a set of all *feasible* solutions, $\mathbf{b}^i$ denotes the best individual in terms of the function *eval* in generation i, $\beta_1, \beta_2 > 1$ and $\beta_1 \neq \beta_2$ (to avoid cycling). In other words, the method decreases the penalty component $W(t+1)$ for the generation $t+1$ if all best individuals in the last k generations were feasible (i.e., in $\mathcal{F}$), and increases penalties if all best individuals in the last k generations were infeasible. If there are some feasible and infeasible individuals as best individuals in the last k generations, $W(t+1)$ remains without change.

[8] Of course, instead of W it is possible to consider a vector of weights $\mathbf{w} = (w_1, \ldots, w_m)$ which are applied directly to violation functions $f_j(\mathbf{x})$. In such a case $penalty(\mathbf{x}) = \sum_{j=1}^{m} w_j f_j(\mathbf{x})$. The discussion in the remaining part of this section can be easily extended to this case.

Third, we could allow self-adaptation of the weight parameter, similarly to the mutation step sizes in the previous section. For example, it is possible to extend the representation of individuals into

$$\langle x_1, \ldots, x_n, W \rangle,$$

where W is the weight. The weight component W undergoes the same changes as any other variable x_i (e.g., Gaussian mutation, arithmetical crossover). However, it is unclear, how the evaluation function can benefit from such self-adaptation. Clearly, the smaller weight W, the better an (infeasible) individual is, so it is unfair to apply different weights to different individuals within the same generation. It might be that a new weight can be defined (e.g., arithmetical average of all weights present in the population) and used for evaluation purpose; however, to our best knowledge, no one has experimented with such self-adaptive weights.

To this end, it is important to note the crucial difference between self-adapting mutation step sizes and constraint weights. Even if the mutation step sizes are encoded in the chromosomes, the evaluation of a chromosome is *independent* from the actual value of σ's. That is,

$$eval(\langle \mathbf{x}, \sigma \rangle) = f(\mathbf{x})$$

for any chromosome $\langle \mathbf{x}, \sigma \rangle$. In contrast, if constraint weights are encoded in the chromosomes, then we have

$$eval(\langle \mathbf{x}, W \rangle) = f_W(\mathbf{x})$$

for any chromosome $\langle \mathbf{x}, W \rangle$. This enables the evolution to 'cheat' in the sense of making improvements by modifying the value of W instead of optimizing f and satisfying the constraints.

3.3 Summary

In the previous subsections we illustrated how the mutation operator and the evaluation function can be controlled (adapted) during the evolutionary process. The latter case demonstrates that not only the traditionally adjusted components, such as mutation, recombination, selection, etc., can be controlled by parameters, but so can other components of an evolutionary algorithm. Obviously, there are many components and parameters that can be changed and tuned for optimal algorithm performance. In general, the three options we sketched for the mutation operator and the evaluation function are valid for any parameter of an evolutionary algorithm, whether it is population size, mutation step, the penalty coefficient, selection pressure, and so forth.

The mutation example of Section 3.1 also illustrates the phenomenon of the *scope of a parameter*. Namely, the mutation step size parameter can have different domains of influence, which we call scope. Using the

$\langle x_1, \ldots, x_n, \sigma_1, \ldots, \sigma_n \rangle$ model, a particular mutation step size applies only to one variable of a single individual. Thus, the parameter σ_i acts on a subindividual level. In the $\langle x_1, \ldots, x_n, \sigma \rangle$ representation the scope of σ is one individual, whereas the dynamic parameter $\sigma(t)$ was defined to affect all individuals and thus has the whole population as its scope.

These remarks conclude the introductory examples of this section; we are now ready to attempt a classification of parameter control techniques for parameters of an evolutionary algorithm.

4. Classification of Control Techniques

In classifying parameter control techniques of an evolutionary algorithm, many aspects can be taken into account. For example:

1 *What* is changed? (e.g., representation, evaluation function, operators, selection process, mutation rate, etc.).

2 *How* the change is made? (i.e., deterministic heuristic, feedback-based heuristic, or self-adaptive).

3 *The scope/level* of change (e.g., population-level, individual-level, etc.).

4 *The evidence* upon which the change is carried out (e.g., monitoring performance of operators, diversity of the population, etc.).

In the following we discuss these items in more detail.

To classify parameter control techniques from the perspective of what is changed, it is necessary to agree on a list of all major components of an evolutionary algorithm (which is a difficult task in itself). For that purpose, assume the following components of an EA:

- Representation of individuals.
- Evaluation function.
- Variation operators and their probabilities.
- Selection operator (parent selection or mating selection).
- Replacement operator (survival selection or environmental selection).
- Population (size, topology, etc.).

Note that each component can be parameterized, and the number of parameters is not clearly defined. For example, an offspring produced

by an arithmetical crossover of k parents $\mathbf{x}_1, \ldots, \mathbf{x}_k$ can be defined by the following formula

$$\mathbf{v} = a_1\mathbf{x}_1 + \ldots + a_k\mathbf{x}_k,$$

where $a_1, \ldots, a_k$, and k can be considered as parameters of this crossover. Parameters for a population can include the number and sizes of subpopulations, migration rates, etc. (this is for a general case, when more then one population is involved). Despite the somewhat arbitrary character of this list of components and of the list of parameters of each component, we will maintain the "what-aspect" as one of the main classification features. The reason for this is that it allows us to locate where a specific mechanism has its effect. Also, this is way we would expect people to search through a survey, e.g., "I want to apply changing mutation rates, let me see how others did it".

As discussed and illustrated in Section 3, methods for changing the value of a parameter (i.e., the "how-aspect") can be classified into one of three categories:

- *Deterministic* parameter control.
  This takes place when the value of a strategy parameter is altered by some deterministic rule. This rule modifies the strategy parameter deterministically without using any feedback from the search. Usually, a time-varying schedule is used, i.e., the rule will be used when a set number of generations have elapsed since the last time the rule was activated.

- *Adaptive* parameter control.
  This takes place when there is some form of feedback from the search that is used to determine the direction and/or magnitude of the change to the strategy parameter. The assignment of the value of the strategy parameter may involve credit assignment, and the action of the EA may determine whether or not the new value persists or propagates throughout the population.

- *Self-adaptive* parameter control.
  The idea of the evolution of evolution can be used to implement the self-adaptation of parameters. Here the parameters to be adapted are encoded into the chromosomes and undergo mutation and recombination. The better values of these encoded parameters lead to better individuals, which in turn are more likely to survive and produce offspring and hence propagate these better parameter values.

This terminology leads to the taxonomy illustrated in Figure 11.1.

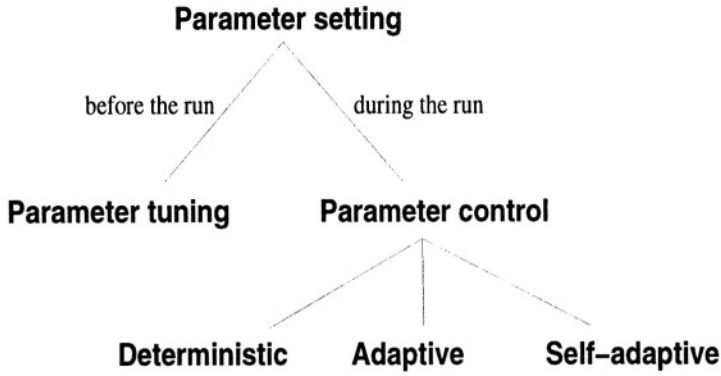


Figure 11.1. Gobal taxonomy of paremeter setting in EAs

Some authors have introduced a different terminology. Angeline (1995) distinguished absolute and empirical rules corresponding to uncoupled and tightly-coupled mechanisms of Spears (1995). Let us note that the uncoupled/absolute category encompasses deterministic and adaptive control, whereas the tightly-coupled/empirical category corresponds to self-adaptation. We feel that the distinction between deterministic and adaptive parameter control is essential, as the first one does not use any feedback from the search process. However, we acknowledge that the terminology proposed here is not perfect either. The term "deterministic" control might not be the most appropriate, as it is not determinism that matters, but the fact that the parameter-altering transformations take no input variables related to the progress of the search process. For example, one might *randomly* change the mutation probability after every 100 generations, which is not a deterministic process. The name "fixed" parameter control might form an alternative that also covers this latter example. Also, the terms "adaptive" and "self-adaptive" could be replaced by the equally meaningful "explicitly adaptive" and "implicitly adaptive" controls, respectively. We have chosen to use "adaptive" and "self-adaptive" for the widely accepted usage of the latter term.

As discussed earlier, any change within any component of an EA may affect a gene (parameter), whole chromosomes (individuals), the entire population, another component (e.g., selection), or even the evaluation function. This is the aspect of the scope or level of adaptation (Angeline, 1995; Hinterding et al., 1997; Smith and Fogarty, 1997; Smith, 1997). Note, however, that the scope/level usually depends on the component of the EA where the change takes place. For example, a change of the mutation step size may affect a gene, a chromosome, or the entire

population, depending on the particular implementation (i.e., scheme used), but a change in the penalty coefficients always affects the whole population. So, the scope/level feature is a secondary one, usually depending on the given component and its actual implementation.

The issue of the scope of the parameter might be more complicated than indicated in Section 3.3, however. First of all, the scope depends on the interpretation mechanism of the given parameters. For example, an individual might be represented as

$$\langle x_1, \ldots, x_n, \sigma_1, \ldots, \sigma_n, \alpha_1, \ldots, \alpha_{n(n-1)/2} \rangle,$$

where the vector α denotes the covariances between the variables $\sigma_1, \ldots, \sigma_n$. In this case the scope of the strategy parameters in α is the whole individual, although the notation might suggest that they act on a subindividual level.

The next example illustrates that the same parameter (encoded in the chromosomes) can be interpreted in different ways, leading to different algorithm variants with different scopes of this parameter. Spears (1995), following (Fogel and Atmar, 1990), experimented with individuals containing an extra bit to determine whether one-point crossover or uniform crossover is to be used (bit 1/0 standing for one-point/uniform crossover, respectively). Two interpretations were considered. The first interpretation was based on a pairwise operator choice: If both parental bits are the same, the corresponding operator is used, otherwise, a random choice is made. Thus, this parameter in this interpretation acts at an individual level. The second interpretation was based on the bit-distribution over the whole population: If, for example 73% of the population had bit 1, then the probability of one-point crossover was 0.73. Thus this parameter under this interpretation acts on the population level. Note, that these two interpretations can be easily combined. For instance, similar to the first interpretation, if both parental bits are the same, the corresponding operator is used. However, if they differ, the operator is selected according to the bit-distribution, just as in the second interpretation. The scope/level of this parameter in this interpretation is neither individual, nor population, but rather both. This example shows that the notion of scope can be ill-defined and very complex. These examples, and the arguments that the scope/level entity is primarily a feature of the given parameter and only secondarily a feature of adaptation itself, motivate our decision to exclude it as a major classification criterion.

Another possible criterion for classification is the evidence used for determining the change of parameter value (Smith and Fogarty, 1997; Smith, 1997). Most commonly, the progress of the search is monitored,

e.g., the performance of operators. It is also possible to look at other measures, like the diversity of the population. The information gathered by such a monitoring process is used as feedback for adjusting the parameters. Although this is a meaningful distinction, it appears only in adaptive parameter control. A similar distinction could be made in deterministic control, which might be based on any counter not related to search progress. One option is the number of fitness evaluations (as the description of deterministic control above indicates). There are, however, other possibilities, for instance, changing the probability of mutation on the basis of the number of executed mutations. We feel, however, that these distinctions are of a more specific level than other criteria and for that reason we have not included it as a major classification criterion.

So the main criteria for classifying methods that change the values of the strategy parameters of an algorithm during its execution are:

1 *What* is changed?

2 *How* is the change made?

Our classification is thus two-dimensional: the type of control and the component of the evolutionary algorithm which incorporates the parameter. The *type* and *component* entries are orthogonal and encompass typical forms of parameter control within EAs. The *type* of parameters' change consists of three categories: deterministic, adaptive, and self-adaptive mechanisms. The *component* of parameters' change consists of six categories: representation, evaluation function, variation operators (mutation and recombination), selection, replacement, and population.

5. Various forms of control

To discuss the experimental efforts of many researchers to control the parameters of their evolutionary algorithms, it it useful to select an ordering principle to group existing work based on *what* is being adapted. In Eiben et al. (1999), we provide a full survey of various forms of control which have been studied by the evolutionary computation community in recent years. In that paper we have selected an ordering principle to group existing work based on *what* is being adapted. Consequently, the discussion corresponded to the earlier list of six components of an EA and we just briefly indicated what the scope of the change is. So, in Eiben et al. (1999), we discussed control of representation, evaluation function, mutation and crossover operators and their probabilities, parent selection, replacement operator, and population.

As we explained in the introduction, 'control of parameters in EAs' includes any change of any of the parameters that influence the action

of the EA, whether it is done by a deterministic rule, feedback-based rule, or a self-adaptive mechanism.[9] Also it is possible to control the various parameters of an evolutionary algorithm during its run. However, most studies considered control of one parameter only (or a few parameters which relate to a single component of EA). This is probably because (1) the exploration of capabilities of adaptation was done experimentally, and (2) it is easier to report positive results in such simpler cases. Combining forms of control is much more difficult as the interactions of even static parameter settings for different components of EA's are not well understood, as they often depend on the objective function (Hart and Belew, 1991) and representation used (Tate and Smith, 1993). Several empirical studies have been performed to investigate the interactions between various parameters of an EA (Eshelman and Schaffer, 1993; Schaffer et al., 1989; Wu et al., 1997). Some stochastic models based on Markov chains were developed and analysed to understand these interactions (Chakraborty et al., 1996; Nix and Vose, 1992; Suzuki, 1993; Vose, 1992).

In combining forms of control, the most common method is related to mutation. With Gaussian mutation we can have a number of parameters that control its operation. We can distinguish the setting of the standard deviation of the mutations (mutation step size) at a global level, for each individual, or for genes (parameters) within an individual. We can also control the preferred direction of mutation.

In evolution strategies (Schwefel, 995), the self-adaptation of the combination of the mutation step-size with the direction of mutation is quite common. Also the adaptation of the mutation step-size occurs at both the individual and the gene level. This combination has been used in EP as well (Saravanan and Fogel, 1994). Other examples of combining the adaptation of the different mutation parameters are given in Yao *et al.* (1997) and Ghozeil and Fogel (1996). Yao *et al.* combine the adaptation of the step size with the mixing of Cauchy and Gaussian mutation in EP. Here the mutation step size is self-adapted, and the step size is used to generate two new individuals from one parent: one using Cauchy mutation and the other using Gaussian mutation; the "worse" individual in terms of fitness is discarded. The results indicate that the method is generally better or equal to using either just Gaussian or Cauchy mutations even though the population size was halved to compensate for generating two individuals from each parent. Ghozeil and

[9]Note that in many papers, the term 'control' is referred to as 'adaptation'.

Fogel compare the use of polar coordinates for the mutation step size and direction over the generally used cartesian representation. While their results are preliminary, they indicate that superior results can be obtained when a lognormal distribution is used to mutate the self-adaptive polar parameters on some problems.

Combining forms of control where the adapted parameters are taken from different components of the EA are much rarer. Hinterding *et al.* (1996) combined self-adaptation of the mutation step size with the feedback-based adaptation of the population size. Here feedback from a cluster of three EAs with different population sizes was used to adjust the population size of one or more of the EAs at 1,000 evaluation epochs, and self-adaptive Gaussian mutation was used in each of the EAs. The EA adapted different strategies for different type of test functions: for unimodal functions it adapted to small population sizes for all the EAs; while for multimodal functions, it adapted one of the EAs to a large but oscillating population size to help it escape from local optima. Smith and Fogarty (1996) self-adapt both the mutation step size and preferred crossover points in a EA. Each gene in the chromosome includes: the problem encoding component; a mutation rate for the gene; and two linkage flags, one at each end of the gene which are used to link genes into larger *blocks* when two adjacent genes have their adjacent linkage flags set. Crossover is a multiparent crossover and occurs at block boundaries, whereas the mutation can affect all the components of a block and the rate is the average of the mutation rates in a block. Their method was tested against a similar EA on a variety of NK problems and produced better results on the more complex problems.

The most comprehensive combination of forms of control is by Lis and Lis (1996), as they combine the adaptation of mutation probability, crossover rate and population size, using adaptive control. A parallel GA was used, over a number of epochs; in each epoch the parameter settings for the individual GAs was determined by using the Latin Squares experiment design. This was done so that the best combination of three values for each of the three parameters could be determined using the fewest number of experiments. At the end of each epoch, the middle level parameters for the next epoch were set to be the best values from the last epoch.

It is interesting to note that all but one of the EAs which combine various forms of control use self-adaptation. In Hinterding *et al.* (1996) the reason that feedback-based rather than self-adaptation was used to control the population size, was to minimize the number of separate populations. This leads us to believe that while the interactions of static parameters setting for the various components of an EA are complex,

the interactions of the dynamics of adapting parameters using either deterministic or feedback-based adaptation will be even more complex and hence much more difficult to work out. Hence it is likely that using self-adaptation is the most promising way of combining forms of control, as we leave it to evolution itself to determine the beneficial interactions among various components (while finding a near-optimal solution to the problem).

However, it should be pointed out that any combination of various forms of control may trigger additional problems related to "transitory" behavior of EAs. Assume, for example, that a population is arranged in a number of disjoint subpopulations, each using a different crossover. If the current size of subpopulation depends on the merit of its crossover, the operator which performs poorly (at some stage of the process) would have difficulties "to recover" as the size of its subpopulation shrinked in the meantime (and smaller populations usually perform worse than larger ones). This would reduce the chances for utilizing "good" operators at later stages of the process.

6. Discussion

The effectiveness of an evolutionary algorithm depends on many of its components, e.g., representation, operators, etc., and the interactions among them. The variety of parameters included in these components, the many possible choices (e.g., to change or not to change?), and the complexity of the interactions between various components and parameters make the selection of a "perfect" evolutionary algorithm for a given problem very difficult, if not impossible.

So, how can we find the "best" EA for a given problem? As discussed earlier, we can perform some amount of parameter tuning, trying to find good values for all parameters before the run of the algorithm. However, even if we assume for a moment that there is a perfect configuration, finding it is an almost hopeless task. Figure 11.2 illustrates this point: the search space S_{EA} of all possible evolutionary algorithms is huge, much larger than the search space S_P of the given problem P, so our chances of *guessing* the right configuration (if one exists!) for an EA are rather slim (e.g., much smaller than the chances of guessing the optimum permutation of cities for a large instance of the traveling salesman problem). Even if we restrict our attention to a relatively narrow subclass, say S_{GA} of classical GAs, the number of possibilities is still prohibitive.[10]

[10]A subspace of *classical* genetic algorithms, $S_{GA} \subset S_{EA}$, consists of evolutionary algorithms where individuals are represented by binary coded fixed-length strings, which has two operators: 1-point crossover and a bit-flip mutation, and it uses a proportional selection.

Note, that within this (relatively small) class there are many possible algorithms with different population sizes, different frequencies of the two basic operators (whether static or dynamic), etc. Besides, guessing the right values of parameters might be of limited value anyway: in this chapter we have argued that any set of static parameters seems to be inappropriate, as any run of an EA is an intrinsically dynamic, adaptive process. So the use of rigid parameters that do not change their values may not be optimal, since different values of parameters may work better/worse at different stages of the evolutionary process.

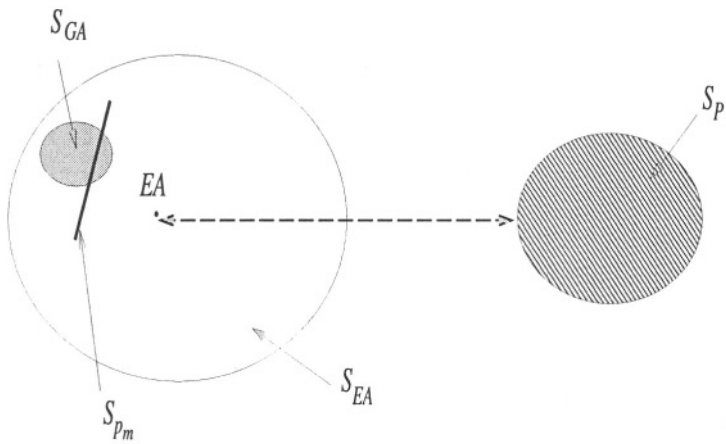


Figure 11.2. An evolutionary algorithm EA for problem P as a single point in the search space S_{EA} of all possible evolutionary algorithms. EA searches (broken line) the solution space S_P of the problem P. S_{GA} represents a subspace of classical GAs, whereas S_{p_m} — a subspace which consists of evolutionary algorithms which are identical except their mutation rate p_m.

On the other hand, adaptation provides the opportunity to customize the evolutionary algorithm to the problem and to modify the configuration and the strategy parameters used while the problem solution is sought. This possibility enables us not only to incorporate domain information and multiple reproduction operators into the EA more easily, but, as indicated earlier, allows the algorithm itself to select those values and operators which provide better results. Of course, these values can be modified during the run of the EA to suit the situation during that part of the run. In other words, if we allow some degree of adaptation within an EA, we can talk about two different searches which take place simultaneously: while the problem P is being solved (i.e., the search space S_P is being searched), a part of S_{EA} is searched as well for the best evolutionary algorithm EA for some stage of the search of S_P.

However, in all experiments reported by various researchers only a tiny part of the search space S_{EA} was considered. For example, by adapting the mutation rate p_m we consider only a subspace S_{p_m} (see Figure 11.2), which consists of all evolutionary algorithms with all parameters fixed except the mutation rate. Similarly, early experiments of Grefenstette (1986) were restricted to the subspace S_{GA} only.

An important objective of this chapter is to draw attention to the potentials of EAs adjusting their own parameters on-line. Given the present state of the art in evolutionary computation, what could be said about the feasibility and the limitations of this approach?

One of the main obstacles of optimizing parameter settings of EAs is formed by the epistasic interactions between these parameters. The mutual influence of different parameters on each other and the combined influence of parameters together on EA behaviour is very complex. A pessimistic conclusion would be that such an approach is not appropriate, since the ability of EAs to cope with epistasis is limited. On the other hand, parameter optimization falls in the category of ill-defined, not well-structured (at least not well understood) problems preventing an analytical approach — a problem class for which EAs usually provide a reasonable alternative to other methods. Roughly speaking, we might not have a better way to do it than letting the EA figuring it out. To this end, note that the self-adaptive approach represents the highest level of reliance on the EA itself in setting the parameters. With a high confidence in the capability of EAs to solve the problem of parameter setting this is the best option. A more sceptical approach would provide some assistance in the form of heuristics on how to adjust parameters, amounting to adaptive parameter control. At this moment there are not enough experimental or theoretical results available to make any reasonable conclusions on the (dis)advantages of different options.

A theoretical boundary on self-adjusting algorithms in general is formed by the no free lunch theorem (Wolpert and Macready, 1997). However, while the theorem certainly applies to a self-adjusting EA, it represents a statement about the performance of the self-adjusting features in optimizing parameters compared to other algorithms for the same task. Therefore, the theorem is not relevant in the practical sense, because these other algorithms hardly exist in practice. Furthermore, the comparison should be drawn between the self-adjusting features and the human "oracles" setting the parameters, this latter being the common practice.

It could be argued that relying on human intelligence and expertise is the best way of drawing an EA design, including the parameter settings. After all, the "intelligence" of an EA would always be limited by the

small fraction of the predefined problem space it encounters during the search, while human designers (may) have global insight of the problem to be solved. This, however, does not imply that the human insight leads to better paremeter settings (see our discussion of the approaches called parameter tuning and parameter setting by analogy in Section 2). Furthermore, human expertise is costly and might not be easily available for the given problem at hand, so relying on computer power is often the most practicable option. The domain of applicability of the evolutionary problem solving technology as a whole could be significantly extended by EAs that are able to configurate themselves, at least partially.

At this stage of research it is unclear just "how much parameter control" might be useful. Is it feasible to consider the whole search space S_{EA} of evolutionary algorithms and allow the algorithm to select (and change) the representation of individuals together with operators? At the same time should the algorithm control probabilities of the operators used together with population size and selection method? It seems that more research on the combination of the types and levels of parameter control needs to be done. Clearly, this could lead to significant improvements to finding good solutions and to the speed of finding them.

Another aspect of the same issue is "how much parameter control is worthwhile"? In other words, what computational costs are acceptable? Some researchers have offered that adaptive control substantially complicates the task of EA and that the rewards in solution quality are not significant to justify the cost (Beasley et al., 1993). Clearly, there is some learning cost involved in adaptive and self-adaptive control mechanisms. Either some statistics are collected during the run, or additional operations are performed on extended individuals. Comparing the efficiency of algorithms with and without (self-)adaptive mechanisms might be misleading, since it disregards the time needed for the tuning process. A more fair comparison could be based on a model which includes the time needed to set up (to tune) and to run the algorithm. We are not aware of any such comparisons at the moment.

On-line parameter control mechanisms may have a particular significance in nonstationary environments. In such environments often it is necessary to modify the current solution due to various changes in the environment (e.g., machine breakdowns, sickness of employees, etc). The capabilities of evolutionary algorithm to consider such changes and to track the optimum efficiently have been studied (Angeline, 1997; Bäck, 1998; Vavak et al., 1996; Vavak et al., 1997). A few mechanisms were considered, including (self-)adaptation of various parameters of the algorithm, while other mechanisms were based on maintenance of genetic diversity and on redundancy of genetic material. These mechanisms

often involved their own adaptive schemes, e.g., adaptive dominance function.

It seems that there are several exciting research issues connected with parameter control of EAs. These include:

- Developing models for comparison of algorithms with and without (self-)adaptive mechanisms. These models should include stationary and dynamic environments.
- Understanding the merit of parameter changes and interactions between them using simple deterministic controls. For example, one may consider an EA with a constant population-size versus an EA where population-size decreases, or increases, at a predefined rate such that the total number of function evaluations in both algorithms remain the same (it is relatively easy to find heuristic justifications for both scenarios).
- Justifying popular heuristics for adaptive control. For instance, why and how to modify mutation rates when the allele distribution of the population changes?
- Trying to find the general conditions under which adaptive control works. For self-adative mutation step sizes there are some universal guidelines (e.g., surplus of offspring, extinctive selection), but so far we do not know of any results regarding adaptation.
- Understanding the interactions among adaptively controlled parameters. Usually feedback from the search triggers changes in one of the parameters of the algorithm. However, the same trigger can be used to change the values of other parameters. The parameters can also directly influence each other.
- Investigating the merits and drawbacks of self-adaptation of several (possibly all) parameters of an EA.
- Developing a formal mathematical basis for the taxonomy for parameter control in evolutionary algorithms in terms of functionals which transform the operators and variables they require.

In the next few years we expect new results in these areas.

References

Angeline, P. (1995). Adaptive and self-adaptive evolutionary computation. In M., P., Y., A., R.J., M., D., F., and T., F., editors, *Computational Intelligence: A Dynamic System Perspective,* IEEE Press, 152-161.

Angeline, P. (1997). Tracking extrema in dynamic environments. In Proceedings of the 6th Annual Conference on Evolutionary Programming, 335-345.

Bäck, T. (1992). The interaction of mutation rate, selection, and self-adaptation within a genetic algorithm. In Männer, R. and Manderick, B., editors, *Proceedings of the 2nd Conference on Parallel Problem Solving from Nature,* North-Holland, 85-94.

Bäck, T. (1992). Self-adaption in genetic algorithms. In Varela, F. and Bourgine, P., editors, *Toward a Practice of Autonomous Systems: Proceedings of the 1st European Conference on Artificial Life,* MIT Press, 263-271.

Bäck, T. (1993). Optimal mutation rates in genetic search. *Proceedings of the 5th International Conference on Genetic Algorithms,* Forrest, S., editor, Morgan Kaufmann, 2-8.

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

Bäck, T. (1998). On the behavior of evolutionary algorithms in dynamic environments. In *Proceedings of the 5th IEEE Conference on Evolutionary Computation,* IEEE Press, 446-451.

Bäck, T., Fogel, D., and Michalewicz, Z., editors (1997). *Handbook of Evolutionary Computation.* Institute of Physics Publishing Ltd, Bristol and Oxford University Press, New York.

Bean, J. and Hadj-Alouane, A. (1992). A dual genetic algorithm for bounded integer programs. Tr 92-53, Department of Industrial and Operations Engineering, The University of Michigan.

Beasley, D., Bull, D., and Martin, R. (1993). An overview of genetic algorithms: Part 2, research topics. *University Computing,* 15(4):170-181.

Chakraborty, U., Deb, K., and Chakraborty, M. (1996). Analysis of selection algorithms: A markov chain approach. *Evolutionary Computation,* 4(2):132-167.

Davis, L. (1989). Adapting operator probabilities in genetic algorithms. In Grefenstette, J., editor, *Proceedings of the 1st International Conference on Genetic Algorithms and Their Applications,* pages 61–69. Lawrence Erlbaum Associates.

De Jong, K. (1975). *The Analysis of the Behavior of a Class of Genetic Adaptive Systems.* PhD thesis, Department of Computer Science, University of Michigan, Ann Arbor, Michigan.

Eiben, A. E., Hinterding, R., and Michalewicz, Z. (1999). Parameter control in evolutionary algorithms. *IEEE Transactions on Evolutionary Computation,* 3(2):124-141.

Eshelman, L. and Schaffer, J. (1993). Crossover niche. *Proceedings of the 5th International Conference on Genetic Algorithms,* Forrest, S., editor, Morgan Kaufmann, 9–14.

Fogel, D. and Atmar, J. (1990). Comparing genetic operators with Gaussian mutations in simulated evolutionary processes using linear systems. *Biological Cybernetics,* 63:111-114.

Ghozeil, A. and Fogel, D. (1996). A preliminary investigation into directed mutations in evolutionary algorithms. *Proceedings of the 4th Conference on Parallel Problem Solving from Nature,* number 1141 in Lecture Notes in Computer Science. Springer, Berlin., 329-335.

Goldberg, D. (1989). *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison-Wesley.

Goldberg, D., Deb, K., and Clark, J. (1992). Genetic algorithms, noise, and the sizing of populations. *Complex Systems,* 6:333-362.

Goldberg, D., Deb, K., and Theirens, D. (1991). Toward a better understanding of mixing in genetic algorithms. *Proceedings of the 4th International Conference on Genetic Algorithms,* Belew, R. and Booker, L., editors, Morgan Kaufmann, 190-195.

Grefenstette, J. (1986). Optimization of control parameters for genetic algorithms. *IEEE Transactions on Systems, Man, and Cybernetics,* 16(1):122-128.

Harik, G., Cantu-Paz, E., Goldberg, D., and Miller, B. (1997). The gambler's ruin problem, genetic algorithms, and the sizing of populations. In Proceedings of the 4th IEEE Conference on Evolutionary Computation, IEEE Press, 7-12.

Hart, W. and Belew, R. (1991). Optimizing an arbitrary function is hard for the genetic algorithms. *Proceedings of the 4th International Conference on Genetic Algorithms,* Belew, R. and Booker, L., editors, Morgan Kaufmann, 190-195.

Hesser, J. and Manner, R. (1991). Towards an optimal mutation probability for genetic algorithms. In Schwefel, H.-P. and Männer, R., editors, *Proceedings of the 1st Conference on Parallel Problem Solving from Nature,* number 496 in Lecture Notes in Computer Science, Springer-Verlag, 23-32.

Hinterding, R., Michalewicz, Z., and Eiben, A. (1997). Adaptation in evolutionary computation: A survey. In Proceedings of the 4th IEEE Conference on Evolutionary Computation, IEEE Press, 65–69.

Hinterding, R., Michalewicz, Z., and Peachey, T. (1996). Self-adaptive genetic algorithm for numeric functions. *Proceedings of the 4th Conference on Parallel Problem Solving from Nature,* number 1141 in Lecture Notes in Computer Science. Springer, Berlin, 420-429.

Joines, J. and Houck, C. (1994). On the use of non-stationary penalty functions to solve nonlinear constrained optimization problems with GAs. In *Proceedings of the 1st IEEE Conference on Evolutionary Computation,* IEEE Press, 579-584.

Lis, J. and Lis, M. (1996). Self-adapting parallel genetic algorithm with the dynamic mutation probability, crossover rate and population size. In Arabas, J., editor, *Proceedings of the 1st Polish National Conference on Evolutionary Computation,* pages 324–329. Oficina Wydawnica Politechniki Warszawskiej.

Michalewicz, Z. and Schoenauer, M. (1996). Evolutionary algorithms for constrained parameter optimization problems. *Evolutionary Computation,* 4:1-32.

Nix, A. and Vose, M. (1992). Modelling genetic algorithms with markov chains. *Annals of Mathematics and Artificial Intelligence,* 5:79-88.

Rechenberg, R. (1973). *Evolutionsstrategie: Optimierung technischer Syseme nach Prinzipien der biologischen Evolution.* Frommann-Holzboog, Stuttgart.

Saravanan, N. and Fogel, D. (1994). Learning strategy parameters in evolutionary programming: An empirical study. In Sebald, A. and Fogel, L., editors, *Proceedings of the 3rd Annual Conference on Evolutionary Programming.* World Scientific.

Schaffer, J., Caruana, R., Eshelman, L., and Das, R. (1989). A study of control parameters affecting online performance of genetic algorithms for function optimization. In Schaffer, J., editor, *Proceedings of the 3rd International Conference on Genetic Algorithms,* Morgan Kaufmann, 51-60.

Schaffer, J. and Morishima, A. (1987). An adaptive crossover distribution mechanism for genetic algorithms. In Grefenstette, J., editor, *Proceedings of the 2nd International Conference on Genetic Algorithms and Their Applications,* Lawrence Erlbaum Associates, 36-40.

Schwefel, H.-P. (995). *Evolution and Optimum Seeking.* Wiley, New York.

Smith, J. (1997). *Self Adaptation in Evolutionary Algorithms.* PhD thesis, University of the West of England, Bristol.

Smith, J. and Fogarty, T. (1996). Adaptively parameterised evolutionary systems: Self adative recombination and mutation in a genetic algorithm. *Proceedings of the 4th Conference on Parallel Problem Solving from Nature,* number 1141 in Lecture Notes in Computer Science. Springer, Berlin, 441-450.

Smith, J. and Fogarty, T. (1997). Operator and parameter adaptation in genetic algorithms. *Soft Computing,* l(2):81-87.

Soule, T. and Foster, J. (1997). Code size and depthe flows in genetic programming. In Koza, J., Deb, K., Dorigo, M., Fogel, D., Garzon,

M,, Iba, H., and Riolo, R., editors, *Proceedings of the 2nd Annual Conference on Genetic Programming,* MIT Press, 313-320.

Spears, W. (1995). Adapting crossover in evolutionary algorithms. In McDonnell, J., Reynolds, R., and Fogel, D., editors, *Proceedings of the 4th Annual Conference on Evolutionary Programming.* MIT Press, 367-384.

Suzuki, J. (1993). A markov chain analysis on a genetic algorithm. *Proceedings of the 5th International Conference on Genetic Algorithms,* Forrest, S., editor, Morgan Kaufmann, 146-153.

Syswerda, G. (1991). Schedule optimization using genetic algorithms. In *Handbook of Genetic Algorithms,* Van Nostrand Reinhold, 332-349.

Tate, D. and Smith, E. (1993). Expected allele coverage and the role of mutation in genetic algorithms. *Proceedings of the 5th International Conference on Genetic Algorithms,* Forrest, S., editor, Morgan Kaufmann., 31-37.

Theirens, D. and Goldberg, D. (1991). Mixing in genetic algorithms. *Proceedings of the 4th International Conference on Genetic Algorithms,* Belew, R. and Booker, L., editors, Morgan Kaufmann, 31-37.

Thierens, D. (1996). Dimensional analysis of allele-wise mixing revisited. *Proceedings of the 4th Conference on Parallel Problem Solving from Nature,* number 1141 in Lecture Notes in Computer Science. Springer, Berlin, 255-265.

Vavak, F., Fogarty, T., and Jukes, K. (1996). A genetic algorithm with variable range of local search for tracking changing environments. *Proceedings of the 4th Conference on Parallel Problem Solving from Nature,* number 1141 in Lecture Notes in Computer Science. Springer, Berlin, 376-385.

Vavak, F., Jukes, K., and Fogarty, T. (1997). Learning the local search range for genetic optimisation in nonstationary environments. In Proceedings of the 4th IEEE Conference on Evolutionary Computation, IEEE Press, 355-360.

Vose, M. (1992). Modeling simple genetic algorithms. In Whitley, L., editor, *Foundations of Genetic Algorithms,* Morgan Kauffmann, 2, 63-74.

Wolpert, D. and Macready, W. (1997). No free luch theorems for optimization. *IEEE Transactions on Evolutionary Computation,* 1(1):67-82.

Wu, A., Lindsay, R., and Riolo, R. (1997). Empirical observation on the roles of crossover and mutation. In Bäck, T., editor, *Proceedings of the 7th International Conference on Genetic Algorithms,* Morgan Kaufmann, 362-369.

Yao, X., Lin, G., and Liu, Y. (1997). An analysis of evolutionary algorithms based on neighbourhood and step sizes. *Proceedings of the 6th Annual Conference on Evolutionary Programming,* number 1213 in Lecture Notes in Computer Science. Springer, Angeline, P., Reynolds, R., McDonnel, J., and Eberhart, R., editors, Berlin, 297-307.

Zhang, B. and Mühlenbein, H. (1995). Balancing accuracy and parsimony in genetic programming. *Evolutionary Computation,* 3(3):17-38.

VI

APPLICATION OF EAS TO PRACTICAL PROBLEMS

This page intentionally left blank

Chapter 12

DESIGN OF PRODUCTION FACILITIES USING EVOLUTIONARY COMPUTING

Alice E. Smith and
Bryan A. Norman

Abstract This chapter discusses the use of genetic algorithms for the design of industrial facilities. Block design of the location of departments or work centers is integrated with the detailed design of aisle structure, material handling system, machine placement and input/output locations. Another aspect is to improve the design optimization objective function by matching the choice of flowpath distance metric with the type of material handling system. A modular approach has been taken where various aspects of this evolutionary facility design framework can be used independently or in conjunction with one another.

Keywords: facility design, genetic algorithms, material handling, flowpaths

1. Introduction

Facility design problems generally involve the partition of a building into departments (work centers or cells) along with a material flow structure and a material handling system to link the departments. The primary objective of the design problem is to minimize the costs associated with production and materials movement over the lifetime of the facility. Such problems occur in many organizations, including manufacturing cell layout, hospital layout (Elshafei, 1977), semiconductor manufacturing, construction site management (Yeh, 1995) and service center layout. They also occur in environmental management of forests, wetlands, etc. (Bos, 1993) By any monetary measure, facilities design is an important problem and one that has assumed even greater importance as manufacturers strive to become more agile and responsive (Tompkins, 1997). For U.S. manufacturers, between 20% to 50% of total operating expenses are spent on material handling and an appropriate

facility design can reduce these costs by at least 10% to 30% (Meller and Gau, 1996). Altering facility designs due to incorrect decisions, forecasts or assumptions usually involves considerable cost, time and disruption of activities. On the other hand, good design decisions can reap economic and operational benefits for a long time.

The problem primarily studied in the literature has been "block layout" which specifies the placement of the departments, without regard for aisle structure and material handling system, machine placement within departments or input/output locations. Block layout is usually a precursor to these subsequent design steps, termed "detailed layout". Two recent survey articles on the block layout problem are by Kusiak and Heragu (1987) and by Meller and Gau (1996). A block layout where departments may have different areas and/or different shapes precludes assigning n departments to m distinct locations as is done in the popular quadratic assignment problem (QAP) formulation of block layout, which requires all departments be of identical shape and size. For unequal area departments, the first formulation appeared in (Armour et al., 1963) as follows. There is a rectangular region, A, with fixed dimensions H and W, and a collection of n required departments, each of specified area a_j and dimensions (if rectangular) of h_j and w_j, whose total area = A = HW. There is a material flow $F_{j,k}$ associated with each pair of departments (j,k) which generally includes a traffic volume in addition to a unit cost to transport that volume. There may also be fixed costs between departments j and k. The mathematical objective is to partition A into n subregions representing each of the n departments, of appropriate area, in order to:

$$\min = \sum_{j=1}^{n} \sum_{\substack{k=1 \\ j \neq k}}^{n} C F_{j,k} d_{j,k,\Pi} \tag{12.1}$$

where C is the cost to transport one unit of flow for one unit of distance and $d_{j,k,\Pi}$ is the distance (using a pre-specified metric, most commonly shortest rectilinear between centroids) between department j and department k in the layout II. They approached this problem by requiring all departments to be made up of contiguous rectangular unit blocks, and then applied departmental adjacent pairwise exchange. Other unit block approaches include Bazaraa (1975), Hassan et al. (1986), Meller and Bozer (1996) and Ziai and Sule (1991). Drawbacks of the unit block approach are that departmental shapes can be impractical and each department must be artificially discretized into blocks.

A general linear programming approach was formulated by Montreuil and others (Montreuil and Ratliff, 1989; Montreuil et al., 1993) to avoid

departmental overlap and minimize interdepartmental flow costs, when first given a "design skeleton". Drawbacks of the mathematical programming approach are: (a) inability to enforce exact departmental areas, (b) rapid increase in the number of variables and constraints as the problem size increases, and (c) dependence on an initial layout skeleton. Related approaches (e.g., Imam and Mir (1993) and Welgama and Gibson (1996)) have the disadvantage that the final layout is a central cluster of departments that may be very different in shape than the normal rectangular bounding area HW. Another approach is to use graph theory to develop the optimal relative locations of the department. Foulds and others (Foulds et al., 1985; Foulds and Robinson, 1976; Foulds and Robinson, 1978) pioneered this approach, however it suffers from a rapid increase in search space size as the design problem size increases and the difficulty in translating the graph to a block layout that resembles a physical facility.

There are other basic formulations for unequal area block layout. One is slicing trees, where the departments and the bounding facility are required to be rectangular and the layout is represented by alternating vertical and horizontal slices(Cohoon and Paris, 1987; Tam, 1992a; Tam, 1992b). A related formulation is the flexible bay structure (Figure 12.1) (Tate and Smith, 1995). This structure first allows slices in a single direction, creating bays, which are then sub-divided into departments by perpendicular slices. Although the flexible bay formulation is slightly more restrictive than the slicing tree formulation, it does allow a natural aisle structure to be inherently created in the layout design, a property that may be useful.

The objective of the work described in this chapter is to provide a comprehensive design framework by integrating both "block layout" (the sizing, shaping and locating of departments or work centers) and "detailed design" (choosing and siting input/output (I/O) stations[1] , aisles, material handling systems (MRS), material flowpaths, flow within departments). The method of representing the planar facility region and its division into departments, cells or work areas is the flexible bay structure (Tate and Smith, 1995). To provide departments that are physically reasonable, a maximum aspect ratio constraint for each department is imposed. The lower this constraint, the more square the departments are forced to be. The solution approach uses one of the most widely applied evolutionary computation techniques, genetic algorithms (GA), which is supplemented with problem specific heuristics. Since a GA is

[1]Also called pick up (P) and delivery (D) points.

an improvement heuristic, even large design problems can be well solved in realistic computation time. Furthermore, the identification of a set (population) of superior solutions, that is inherent in a GA, could be useful for facility design.

Basic assumptions made for this chapter are:

- This is a greenfield (new) facility.
- The departments and the facility are rectangular and planar.
- The department areas are fixed and sum to the facility area (equivalently, excess facility area is included as "dummy" departments).
- Aisles have negligible area compared to the facility area.
- Aisle capacity and direction of flow are not considered (i.e., two way flow through each aisle is allowed).
- Material flow occurs between department I/O points around the perimeters of departments.
- Projected material flow volumes and unit cost of material handling are known with certainty.

2. Design for Material Flow When the Number of I/O Points is Unconstrained

The first step is to explicitly consider the material flow paths as the block layout is designed. To do this block layouts are constructed using the flexible bay representation introduced in (Tate and Smith, 1995). However, the layouts were evaluated by assuming that material must move along the perimeters of the departments in the layout and not through the middle of them. This distance metric is known as the contour distance (Norman et al., 1998; Norman et al., 2001; Sinriech and Edouard, 1996; Sinriech and Tanchoco, 1995; Sinriech and Tanchoco, 1997). The contour distance represents a more realistic estimate of how material moves along aisles through an actual facility than either the rectilinear or Euclidean distance metrics that run between department centroids. Initially, it is assumed that each department can have an unconstrained number of material entry and exit points and block layouts are constructed to minimize the total material travel distance.

Due to the structure of the contour measure the set of candidate I/O points for a department can be limited to the locations where that department intersects the corner of any of its adjacent departments, as

proven in (Norman et al., 2001). Using the example of Figure 12.1, the candidate I/O points include all of the shaded circles in the left portion of the figure. To clarify the contour distance measure, if the I/Os used were as shown on the right, the material will traverse over the perimeters shown by the dashed line.

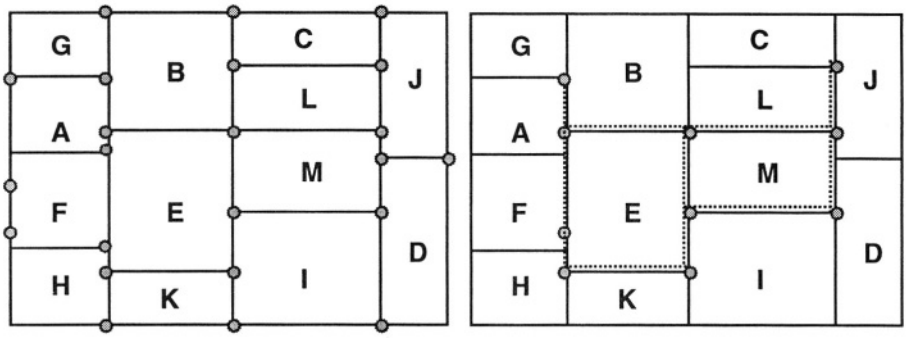


Figure 12.1. A flexible bay block layout with candidate I/O points (left) and implied flowpaths for selected I/O points (right)

When each department can have an unconstrained number of I/O stations the interdepartmental aisle travel distances can be found by formulating the problem as one of finding the shortest path on a network. All of the arc lengths in the resulting shortest path problem will be positive since they represent physical distances. The shortest path problem with positive arc lengths has been well studied in the network optimization literature and efficient algorithms exist for solving this problem (Ahuja et al., 1993). This makes it possible to quickly evaluate the actual aisle travel distance for each layout that is generated during the search process. The objective function of the integrated optimization is:

$$Z(\Pi) = \sum_{i=1}^{n} \sum_{\substack{j=1 \\ j \neq i}}^{n} CF_{i,j} d_{i,j} + m^3 (Z_{feas} - Z_{all}) \qquad (12.2)$$

where m is the number of departments in layout II which violate the aspect ratio constraint, Z_{feas} is the objective function value for the best feasible solution found so far, and Z_{all} is the unpenalized objective function value for the best solution found so far. The penalty function is a variation of the adaptive one by Smith and others (Coit and Smith; Coit et al., 1996; Smith and Tate, 1993). In this case $d_{i,j}$ is defined as

the shortest rectilinear distance along departmental contours between the I/O stations of departments i and j as given by:

$$d_{i,j} = \min \{d_{i,j,k,l} : \ k \in Loc_i, \ l \in Loc_j\} \tag{12.3}$$

where Loc_i and Loc_j are the set of candidate I/O locations for departments i and j, respectively, and $d_{i,k,j,l}$ is the shortest distance, following the department contours, from candidate I/O location k of department i to candidate I/O location l of department j.

The block layout GA works with a variable length encoding of the layout where there is a one to one correspondence between each encoding and each layout. The encoding is a permutation of departments that specifies their order within the layout, with a concatenated string indicating where the bay breaks within the permutation occur. For example, the flexible bay layout of Figure 12.1 would be represented with the following encoding:

G A F H B E K C L M I J D 4 7 11

where the last three characters indicate bay breaks after departments H, K and I.

The algorithm begins with a randomly generated initial population of chromosomes. In each generation, crossover and mutation are applied to create new solutions. Solutions are selected for crossover using a rank-based quadratic method and the worst solutions are deleted during each generation to maintain a constant population size. Tate and Smith (1995) includes the details of these. Crossover is accomplished through a variant of uniform crossover, where two parents create one offspring. The department sequence is subject to uniform crossover with repair to ensure feasible permutations. The bay structure is taken directly from one parent or another with equal probability. The child is evaluated and the worst solution from the current population is deleted and the child solution is added to the population.

Crossover and mutation are performed independently of each other. Mutation is performed by randomly selecting members of the current population with both parents and offspring each having an equal probability of selection. Mutation consists of either permutation altering (50%), or adding (25%) or deleting (25%) a bay. The permutation mutation is inversion between two randomly selected departments. The mutated solutions are evaluated and replace the worst solutions in the current population.

Each new solution that is generated using crossover or mutation is evaluated using the following evaluation procedure:

1 Determine the current facility design from the chromosome

2 Calculate the number of infeasible departments regarding the aspect ratio constraint

3 Construct a network of nodes corresponding to each department in the design and its candidate I/O locations

4 Find the shortest path between each pair of nodes in the network

5 Sum the flow costs along the shortest path according to equation 2

The GA settings were the same as in (Tate and Smith, 1995): population size of 10, mutation rate of 50% (5 mutants per generation) and number of generations generated = 100,000.

Figure 12.2 compares the best Tate and Smith (1995) layout for the classic 20 department Amour and Buffa problem (Armour et al., 1963) using an aspect ratio constraint of 3 to the best one found using the contour distance based algorithm of this section. The implicit centroid to centroid aisles result in parallel paths that traverse departments, and traffic through departments that would normally be unacceptable in actual facilities while the contour distance metric with the selected I/Os is much closer to a workable physical design. More results can be found in (Norman et al., 2001).

3. Design for Material Flow for a Single I/O Point

A logical extension to the ideas presented in the previous section is to consider the case where there are a limited number of I/O points for each of the departments. First, a single I/O point per department is considered by optimizing the I/O placement for a given block layout. This problem can be modeled as a Mixed Integer Programming (MIP) problem. Problem instances with 14 or fewer departments can be solved with standard solvers such as CPLEX (1999). Unfortunately, the problems do not scale up readily so it is necessary to use heuristic methods to solve larger problems. Two classes of heuristics have been developed (Arapoglu et al., 1999) and these are now described in more detail.

The first is a constructive greedy heuristic that temporarily assigns flows to different I/O locations in the departments based on the data in the from-to chart. The I/O location for one department is fixed and the problem is resolved. This process is repeated until all of the departments' I/O locations are fixed. This solution is transformed to a local optimum by perturbing the I/O locations for each department considering one department at a time. The best perturbation is noted and the layout

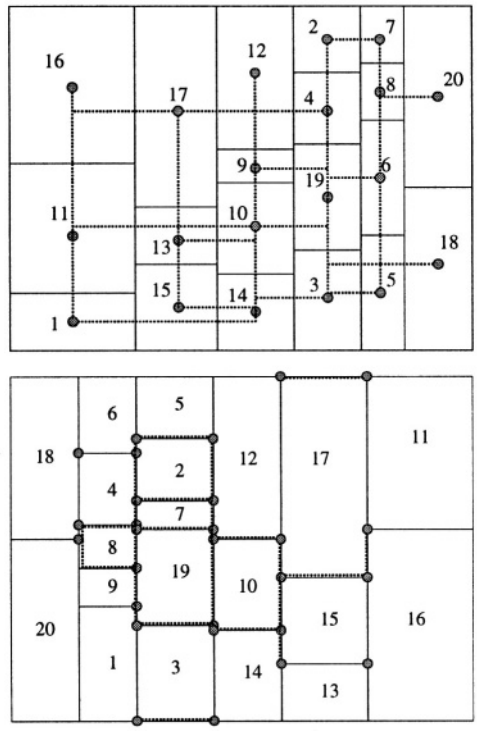


Figure 12.2. I/O points and implied flow paths (dashed) for the centroid rectilinear distance measure (top) and the contour distance measure (bottom) for the Armour and Buffa problem with a maximum aspect ratio constraint of 3.

is modified accordingly. This perturbation procedure is repeated and it stops when there is no more improvement possible. The entire procedure is quite fast and only requires seconds for problems with 50 departments.

The second heuristic utilizes a GA methodology to determine near-optimal I/O assignments. A permutation encoding is used to represent a given allocation of I/O locations to the departments. Each candidate I/O location is numbered (to a maximum of $2N - 2$) and it is possible that two neighboring departments might use the same *location* as their I/O. In that case the same I/O number is used to represent both I/Os. The example below denotes the encoding corresponding to the layout given in Figure 12.3 using the I/O numbering specified on the figure.

Department	A	B	C	D	E	F	G	H	I	J	K	L	M
I/O Locations	3	10	15	18	11	5	2	6	12	15	12	16	10

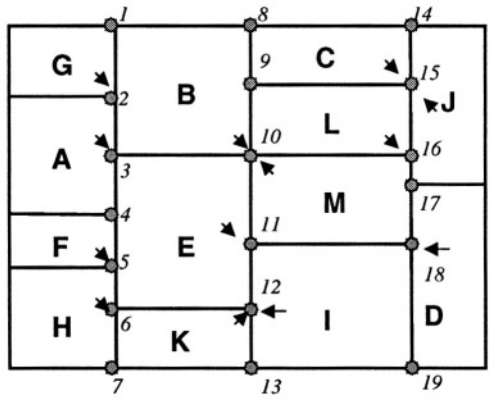


Figure 12.3. Candidate I/O points arbitrarily numbered with one selected for each department.

The fitness of a given solution is the cost of the material flow between each pair of departments using the selected I/O locations and the contour distance metric.

The GA mechanisms were chosen based on experimentation, and although this set provides superior performance, many other GA parameters and evolution mechanisms will provide similar performance. From the old generation of 50, one parent is selected using a tournament of size 2 while the other parent is selected randomly from the population (excluding the parent already selected via the tournament). Biased uniform crossover with probability of 1.0 is applied to produce one offspring, with a preference of 0.70 for alleles from the fitter parent. Note that every crossover produces a feasible allocation of I/O locations and thus, there was no repair needed. This continues until 50 offspring are created each generation.

The offspring are subject to mutation with a probability of 0.5. Every allele is tested for mutation individually and independently with a preselected mutation probability of 0.10. If an allele is chosen for mutation, the new allele is selected randomly from the feasible I/O locations for that department, not including the current value. Again, every mutation results in a feasible allocation of I/O locations. A wide variety of combinations of mutation probabilities were tried with the result that the GA is very robust from mutation probabilities ranging from 0.05 to 1.0, however 0.5 is marginally better than the others.

The 50 mutated offspring are combined with 10 elite solutions from the old population and sorted. The top 50 solutions are kept to form the

new population. The GA is terminated after 500 consecutive generations without any improvement in the best objective function value.

The relaxed MIP lower bound, the objective function results of the IP, the constructive heuristic, and a summary of five runs GA are compared in Table 12.1 for four versions of the Armour and Buffa (Armour et al., 1963) 20 department problem. The GA was very consistent with all seeds finding the optimal solution for each problem instance with a known optimal solution except for the first instance. The best solution of the GA equals or betters the constructive heuristic in every case, although the amount of improvement varies.

A comparison of computational effort is germane. In Table 12.1, the CPU times in seconds on a Sun Ultra Enterprise-2 workstation with dual 200Mhz Sparc CPUs are presented for the constructive heuristic and one run of the GA. The IP, of course, took considerably more time for each problem instance. The GA required about 50 times longer than the constructive heuristic. A strategy balancing computational effort with solution quality would use the constructive heuristic as an optimization subroutine for most of the block layout design optimization and use the GA as the optimization subroutine for the best few solutions of the population in the later generations.

Table 12.1. Results Comparing GA with Constructive Heuristic.

Problem	**Lower** Bound	**Integer** Program	Const Heur.	**Best** GA	**GA** CV	Const Heur. CPU(s)	GA CPU(s)
A&B 1	223.17	344.82	357.02	345.89	0.01	0.14	9.61
A&B 2	202.19	334.94	345.68	334.94	0.00	0.17	7.95
A&B 3	391.39	461.42	497.32	461.42	0.00	0.20	8.20
A&B 4	315.47	441.87	490.45	441.87	0.00	0.27	9.42

4. Considering Intradepartmental Flow

In addition to constraining the number of I/O points in each department it is also important to consider any intradepartmental flow patterns that may exist. A first step, as described in detail in (Norman et al., 2000), is to consider designs where there are separate input (I) locations and output (O) locations. Consider departments E, G and N in Figure 12.4 where each exhibits a common type of intradepartmental flow pattern. Department E has a *U-shaped* flow, department G has a *linear* flow and department N has a *circular* flow. However, in departments E and G the input point for the department is distinct from the output

point. Modeling this situation as one or two I/O locations would be incorrect and could lead to designs that would not perform well in practice. It may also be necessary to include constraints that require that the I location be on the opposite (or same) side of the department as the O location as would arise in a linear (or U-shaped) layout. Little work has previously been done investigating these intradepartmental flow patterns but the methodology presented above can be adapted to handle this additional consideration by adding constraints on the number and location of I/Os for each department.

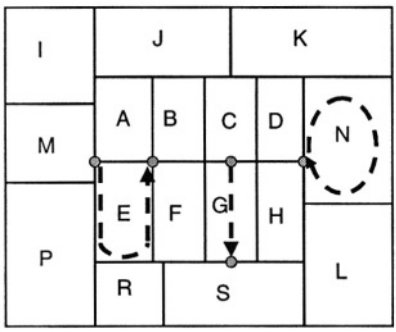


Figure 12.4. U,L and C type intradepartment flows.

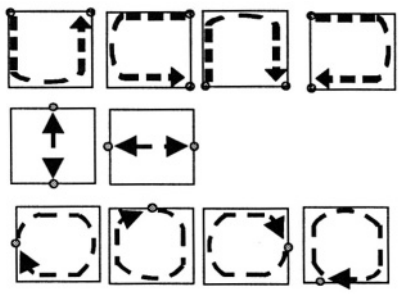


Figure 12.5. Variations of U, L and C flows.

Assumptions made in this section are that department locations have already been determined (the block layout is chosen) and that the flow pattern type for each department is known. It is assumed that for a U-shaped flow, material enters the department at one corner and exits at another corner. Given this assumption, there are eight different U-shaped flow possibilities. The first row of Figure 12.5 shows four of these and the other four can be constructed by reversing the flow directions on each diagram. In a similar manner it is assumed that the linear flows go from the midpoint of one side of the department to the midpoint of the opposite side of the department as indicated in the second row of Figure 12.5. Flows in a circular flow pattern enter and exit at the same location as indicated in the third row of Figure 12.5. Theoretically, the I/O location for a circular flow pattern could be anywhere on the department perimeter but using the results discussed in section 2 only the four corners and all locations where the department intersects either the corner or midpoint of one side of any other department need to be considered.

Potential I and O sites are numbered in a structured way. For any department, point 1 refers to the bottom left corner of the department, point 2 refers to the midpoint of the left side, point 3 refers to the top left corner, points 4 and 5 refer to the midpoints of the bottom and top edges respectively, and points 6, 7, and 8 refer to the bottom, midpoint, and top of the right side of the department. Points 9 and greater refer to any additional points where the department intersects either the midpoints or corner points of a side of another department. The numbering for points greater than or equal to 9 begins on the left side of the department, then continues to the bottom of the department, followed by the top and then the right side of the department. For a U-shaped flow the I/O points can only be at locations 1, 3, 6, and 8 because these represent the four corner points (note only certain pairs represent feasible U-shaped flows). For a linear flow the I/O points can only be at locations 2, 4, 5, and 7 because these represent the midpoint locations for each side. For a circular flow the I/O point can be at any of the locations 1 to 8 or at locations 9 to p_i where p_i is dependent on the physical layout of the departments. The number of candidate locations can be reduced if the flow patterns for the departments adjacent to department i are known.

Once all candidate I and O locations have been determined for all departments, the material flow network can be constructed. This network consists of nodes at each department's candidate I and O locations and arcs consisting of the departmental edges connecting these nodes. It is possible to determine the distance between the I and O locations of each pair of departments by solving a shortest path problem on the underlying network (Ahuja et al., 1993). These distances represent the contour distances for moving material between departments.

In the GA, chromosomes consist of genes that carry the necessary information about the I/O points for each department. A chromosome has n genes for a layout with n departments where each gene corresponds to a department. Shown below is an example chromosome with five genes corresponding to the I/O pairs for each of five departments in a layout.

Chromosome (4,4; 6,1; 3,8; 5,4; 7,2)

Department	1	2	3	4	5
Input	4	6	3	5	7
Output	4	1	8	4	2

The first department has its I point at node 4 in the layout and its O point at node 4 in the layout. Thus, any flow to department one will enter via node four and will exit via node four. The other genes have

the same structure and meaning. It is necessary to select I/O points in such a way that they form a feasible pair by representing one of the three flow pattern types. For example, the pair (1,2) would never be permitted since this does not represent any of the flow pattern types.

Creating a set of feasible lists for each flow pattern type solves the feasibility problem that may arise when creating random I/O pair assignments. These lists are used in the GA when there is a need to create a random I/O assignment. For example, if department 2 has an L flow pattern type then it can only have one of the four I/O assignments shown in Figure 12.5. Note that the number of C candidate I/O assignments will vary from department to department.

A population size of 150 was used as was an elitist strategy where the best 30% are copied exactly from one generation to the next. The GA uses the same selection and biased uniform crossover earlier described. The rate of mutation is governed by two parameters. First, 10% of the population undergo mutation. Second, in a member of the population that is selected for mutation, 10% of the alleles are changed. If the I/O pair is changed for departments with flow type C or L the new I/O pair is randomly generated from the candidate list of I/O pairs and is required to be different from the current I/O pair. If a department with flow type U is mutated then either I or O, or both, are changed. The mutated chromosome replaces the original chromosome in the new population.

Two layouts that utilize the flow data from the 20 department problem of Armour and Buffa (Armour et al., 1963) were used to create 12 test problems of single and mixed intradepartment flows. Results, over 10 runs of each problem, shown in Table 12.2 indicate that the GA provides optimal or near optimal solutions regularly. Each GA run took less than 25 seconds of CPU time on a Sun Spare 20 workstation.

5. Material Handling System Design

Another aspect of improving facility design using an evolutionary approach is the choice of distance metric. This should depend on the material handling system (MHS). The movement of an overhead crane is best measured by the Tchebyshev distance metric (the maximum of the movement in either the x or the y direction). However, fork trucks and AGVs have to follow the aisle structure traveling around departments and should be modeled using the contour distance metric. For other MHS, such as conveyors, a Euclidean distance metric is most appropriate.

As in the work earlier in this chapter, it has been generally implicitly assumed that the unit handling costs and the distance metric are the

Table 12.2. Results from intradepartmental flow test problem.

Problem	Flow Pattern	Optimal or Best Known	GA		
			MIN	MAX	AVG
AB-20-1	All C	344.8 *	347.5	357.8	350.7
AB-20-1	All U	440.1	440.1	454.9	445.4
AB-20-1	All L	673.8	673.8	681.9	676.7
AB-20-1	Mixed 1	507.2	507.2	511.3	508.6
AB-20-1	Mixed 2	489.5	489.5	495.8	492.4
AB-20-1	Mixed 3	477.8	478.3	484.4	479.9
AB-20-2	All C	334.9 *	335.1	344.5	339.5
AB-20-2	All U	432.2	432.2	434.0	432.4
AB-20-2	All L	712.6	712.6	719.0	716.8
AB-20-2	Mixed 1	518.2	518.2	525.2	520.3
AB-20-2	Mixed 2	535.2	535.2	541.9	536.3
AB-20-2	Mixed 3	543.0	544.3	549.7	546.0

* indicates the optimal solution was verified using CPLEX

same for all flows within a facility. But in real-life problems, handling costs differ for different MH devices, and similarly, different materials and departments are best served by different MH systems. In the novel formulation of (Smith et al., 2000) the total material handling cost for a block layout is expressed as follows:

$$\text{Total Cost} = \sum_{k=1}^{m} \sum_{i}^{n} \sum_{j \neq i}^{n} [d_{kij} F_{ij} U_{kij} + I_{kij}] \tag{12.4}$$

where,

d_{kij} distance between departments i, j using the prespecified metric for MHS type k
F_{ij} volume of material flow between departments i, j
U_{kij} unit handling cost when MHS type k is used
I_{kij} first cost when MHS type k is used
n number of departments
m number of types of MHS

This optimization problem was solved with the same GA that was introduced in section 2 with a population size of 20 and a mutation rate of 0.5. Selection, crossover and mutation are the same as in section 2.

Test problems were adapted from the well-known 10-department van Camp test problem (van Camp et al., 1991) using an aspect ratio con-

straint of 3. The flow volumes between departments are taken from original test problem and a distance metric (material handling device) was randomly assigned for each flow according to three sets of probabilities specified below:

- 50% Euclidean, 25% rectilinear, 25% Tchebyshev
- 25% Euclidean, 50% rectilinear, 25% Tchebyshev
- 25% Euclidean, 25% rectilinear, 50% Tchebyshev

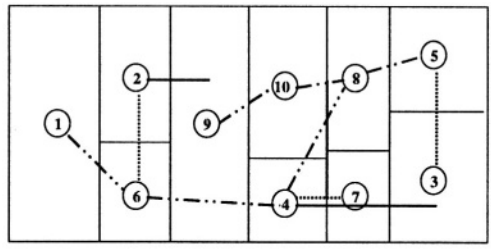
First and unit costs were developed tor each MHS. tor Euclidean, these are 75 and 1.2, respectively. For rectilinear, they are 15 and 1.5, respectively and for Tchebyshev, they are 100 and 0.75, respectively. The best layouts over ten random number seeds are shown in Figure 12.6. Note that the type of MHS strongly affects block layout and flowpaths. Also note that the distance metrics assume movement between department centroids. A next step is to link this idea of heterogeneous MHS with the contour distance metric and I/O placement methods discussed in the previous sections.

6. Concluding Remarks

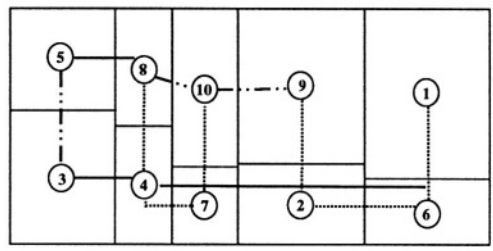
Facility design is a difficult problem and one that has not received enough support from the literature, primarily because of the computational challenges. With the advent of powerful metaheuristics coupled with great improvements in computational speed and capacity, a new generation of analytic tools for facility design is possible. This chapter has described several methods for formulating and solving more realistic versions of the facility design problem. Each method includes evolutionary computation as a primary optimisation method. These approaches can be utilised together or individually depending on the problem to be solved and the amount of computational effort allowed. These approaches allow for an integration of block and detailed design, and inclusion of the important factors of flow within departments and material handling device used for each flow.

Acknowledgments

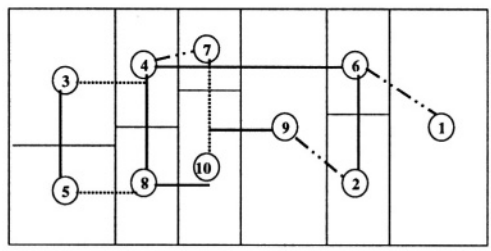
The authors gratefully acknowledge U.S. National Science Foundation grant DMI-990832 for supporting this research.



50% E, 25% R, 25% T.



25% E, 50% R, 25% T.



25% E, 25% R, 50% T.

Figure 12.6. Layouts when considering different material handling devices within a single facility.

References

Ahuja, R. K., Magnanti, T. L. and Orlin, J. B. (1993). *Network Flows: Theory, Algorithms, and Applications.* Prentice-Hall, Upper Saddle, NJ.

Armour, G. C. and Buffa, E. S. (1963). A heuristic algorithm and simulation approach to relative allocation of facilities. *Management Science,* 9, 294-309.

Arapoglu, R. A., Norman, B. A. and Smith, A. E. (1999). Locating Input and Output Points in Facilities Design: A Comparison of Constructive, Evolutionary and Exact Methods. Technical Report, 99-1, Dept. of Industrial Engineering, University of Pittsburgh, to appear *IEEE Transactions on Evolutionary Computation.*

Bazaraa, M. S. (1975). Computerized layout design: A branch and bound approach. *AIIE Transactions,* 7(4), 432-438.

Bos, J. (1993). Zoning in forest management: a quadratic assignment problem solved by simulated annealing. *Journal of Environmental Management,* 37, 127-145.

Cohoon, J. P. and Paris, W. D. (1987). Genetic placement. *IEEE Transactions on Computer-Aided Design,* 6, 956-964.

Coit, D. W. and Smith, A. E. (1996). Penalty guided search for reliability design optimization. *Computers and Industrial Engineering,* 40, 895-904.

Coit, D. W., Smith, A. E. and Tate, D. M. (1996). Adaptive penalty methods for genetic optimization of constrained combinatorial problems. *INFORMS Journal on Computing,* 8, 173-182.

CPLEX Optimization Inc.. CPLEX 6.0 User Documentation, 1999, web: www.cplex.com.

Elshafei, A. N. (1977). Hospital layout as a quadratic assignment problem. *Operational Research Quarterly,* 28, 167-179.

Foulds, L. R., Gibbons, P. B. and Giffin, J. W. (1985). Facilities layout adjacency determination: An experimental comparison of three graph theoretic heuristics. *Operations Research,* 33, 1091-1106.

Foulds, L. R. and Robinson D. F. (1976). A strategy for solving the plant layout problem. *Operational Research Quarterly* 27, 845-855.

Foulds, L. R. and Robinson D. F. (1978). Graph theoretic heuristics for the plant layout problem. *International Journal of Production Research,* 16, 27-37.

Hassan, M. M. D., Hogg, G. L. and Smith, D. R. (1986). Shape: A construction algorithm for area placement evaluation. *International Journal of Production Research,* 24, 1283-1295.

Imam, M. H. and Mir, M. (1993). Automated layout of facilities of unequal areas. *Computers and Industrial Engineering,* 24, 355-366.

Kusiak, A. and Heragu, S. S. (1987). The facility layout problem. *European Journal of Operational Research,* 29, 229-251.

Meller, R. D. and Bozer, Y. A. (1996). A new simulated annealing algorithm for the facility layout problem. *International Journal of Production Research,* 34, 1675-1692.

Meller, R. D. and Gau, K.-Y. (1996). The facility layout problem: Recent and emerging trends and perspectives. *Journal of Manufacturing Systems,* 15, 351-366.

Montreuil, B. and Ratliff, H. D. (1989). Utilizing cut trees as design skeletons for facility layout. *IIE Transactions,* 21, 136-143.

Montreuil, B., Venkatadri, U. and Ratliff, H. D. (1993). Generating a layout from a design skeleton. *IIE Transactions,* 25, 3-15.

Norman, B. A., Smith, A. E. and Arapoglu, R. A. (1998). Integrated facilities layout using a perimeter distance metric. *Parallel Problem Solving from Nature (PPSN V)* (A. E. Eiben, T. Baeck, M. Schoenauer and H.-P. Schwefel, editors), Lecture Notes in Computer Science 1498, Springer-Verlag, Berlin, Germany, 937-946.

Norman, B. A., Smith, A. E. and Arapoglu, R. A. (2001). Integrated facilities layout using a perimeter distance measure. *IIE Transactions,* 33, 337-344.

Norman, B. A., Smith, E. A., Yildirim, E. and Tharmmaphornphilas, W. (2000). An Evolutionary Approach to Incorporating Intradepartmental Flow into Facilities Design. Technical Report 00-02, Dept. of Industrial Engineering, University of Pittsburgh, 2000, to appear *Advances in Engineering Software.*

Sinriech, D. and Edouard, S. (1996). A genetic approach to the material flow network design problem in SFT based systems. *Proceedings of the Fifth IE Research Conference,* 411-416.

Sinriech, D. and Tanchoco, J. M. A. (1995). An introduction to the segmented flow approach for discrete material flow systems. *International Journal of Production Research,* 33, 3381-3410.

Sinriech, D. and Tanchoco, J. M. A. (1997). Design procedures and implementation of the segmented flow topology (SFT) for discrete manufacturing systems. *HE Transactions,* 29, 323-335.

Smith, A. E., Ozdemir, G., and Norman, B. A. (2000). Explicitly incorporating multiple material handling systems within block layout design. *Material Handling Research Colloquium, Material Handling Institute of America,* June 2000, CD Rom format.

Smith, A. E. and Tate, D. M. (1993). Genetic optimization using a penalty function. *Proceedings of the Fifth International Conference on Genetic Algorithms,* 499-505.

Tarn, K. Y. (1992a). Genetic algorithms, function optimization, and facility layout design. *European Journal of Operational Research,* 63, 322-346.

Tam, K. Y. (1992b). A simulated annealing algorithm for allocating space to manufacturing cells. *International Journal of Production Research,* 30, 63-87.

Tate, D. M. and Smith, A. E. (1995). Unequal-area facility layout by genetic search. *IIE Transactions,* 27, 465-472.

Tompkins, J. A. (1997). Facilities planning: A vision for the 21st century. *IIE Solutions,* August 1997, 18-19.

van Camp, D. J., Carter, M. W. and Vannelli, A. (1991). A nonlinear optimization approach for solving facility layout problems. *European Journal of Operational Research,* 57, 174-189.

Welgama, P. S. and Gibson, P. R. (1996). An integrated methodology for automating the determination of layout and materials handling system. *International Journal of Production Research,* 34, 2247-2264.

Yeh, I.-C., (1995). Construction-site layout using annealed neural network. *Journal of Computing in Civil Engineering,* 9, 201-208.

Ziai, M. R. and Sule, D. R. (1991). Computerized facility layout design. *Computers and Industrial Engineering,* 21, 385-389.

This page intentionally left blank

Chapter 13

VIRTUAL POPULATION AND ACCELERATION TECHNIQUES FOR EVOLUTIONARY POWER FLOW CALCULATION IN POWER SYSTEMS

Kit Po Wong and
An Li

Abstract This paper first introduces the concept of virtual population for the formation of high quality chromosomes or individuals in a population. It then describes the numerical acceleration and analytical acceleration techniques for the creation of the virtual population. The new concept and the developed acceleration techniques are embedded into the standard GA algorithm to form the Accelerated Genetic Algorithm (AGA). The power and usefulness of the new concept and techniques are demonstrated through the applications of AGA to solving the Branin RCOS, De Jong 1 and Colville problems. The load flow problem in power systems is then introduced. The new techniques developed are incorporated in a constrained genetic algorithm based load flow algorithm. The enhanced algorithms are then applied to solving the load flow problem of the Klos-Kerner power system under very heavy-load condition.

Keywords: genetic algorithm, optimisation, power system, load flow

1. Introduction

Evolutionary computation offers several algorithms for solving optimisation problems. These algorithms consist of Genetic Algorithms (GAs), Evolutionary Programming (EP) and Evolutionary Strategies (ES) (Holland, 1975; Goldberg, 1989). These approaches are adaptive and have the ability to determine the global optimum solution. They can be taken as search techniques based on an analogy with biology in which a group or population of solutions evolves generation by generation through natural selection. In their implementations, for the GA case for example,

a population of candidate solutions, which are referred to as chromosomes, evolves to an optimum solution through the operation of genetic operators consisting of reproduction, crossover, and mutation.

There are many factors affecting the robustness and speed of the evolutionary optimisation algorithms. A basic factors are the size of the population of chromosomes in GA or individuals in EP and the number of generations. However, when the population size is increased more than its minimum size, the computation time for the evolution process to find the optimum solution can be increased if the number of generations required for obtaining the optimal solutions cannot be reduced sufficiently. Here the minimum population size is defined as the minimum size required by the algorithm to find the global optimum solution for any number of trials. The minimum population size can be reduced by forming high quality chromosomes in a population.

This paper first introduces the concept of virtual population for the formation of high quality chromosomes in a population. It then develops two kinds of solution acceleration techniques, namely numerical and analytical acceleration techniques, for forming and processing the virtual population. While the new concept and the acceleration techniques are applicable to all the evolutionary optimisation algorithms, they are, in this paper, embedded into the standard GA algorithm to form the Accelerated Genetic Algorithm (AGA). The effectiveness and power of the new concept and techniques are demonstrated through the applications of AGA to the Branin RCOS, De Jong 1 and Colville problems (Michalewicz, 1996).

Turning to the operation and planning of power systems, it is required to determine the voltage profile and power flow in the network system under some specified loading conditions. This problem is called the load flow or power flow problem and is a very fundamental problem for power system operation and planning engineers. It is well-known that under very heavy-load condition, conventional method based on Newton-Raphson method can fail to find the load flow solution due to the singularity of the system Jacobian matrix. By treating the load flow problem as an optimisation problem, the authors have developed a constrained GA load flow (CGALF) (Wong et al., 1997) algorithm, which is capable of determining the load flow solution when the power system is operating at its ceiling operation point. However, its computational efficiency needs to be greatly improved.

To enhance the robustness and reduce the computational requirement of CGALF, the solution acceleration techniques developed in this paper are incorporated into this algorithm to form two algorithms CGALFA and CGALFB. For the purposes of demonstrating and comparing the

power and the performance of CGALFA and CGALFB, application studies performed on the Klos-Kerner 11-node system are presented in the paper. The Klos-Kerner test systems are adopted in the present work because it is a practical system and is of particular interest when it is under the heavy-load condition. CGALFB has been found to be very powerful, robust and efficient.

2. Concept of Virtual Population

The standard GA will be taken as the reference in the following sections. A virtual population consists of the current population of candidate solution chromosomes and a number of new populations derived from the current population as shown in Fig. 13.1. The use of the virtual population will enable a new population with sufficient diversity to be formed particularly in the early stage of the evolutionary process. This offers an efficient way to prevent premature convergence from happening. In Fig. 13.1, the new populations *A* to *N* are derived from the current population by solution acceleration techniques to be developed. From the populations in the virtual population, chromosomes are to be selected to form the resultant population as shown in Fig. 13.1.

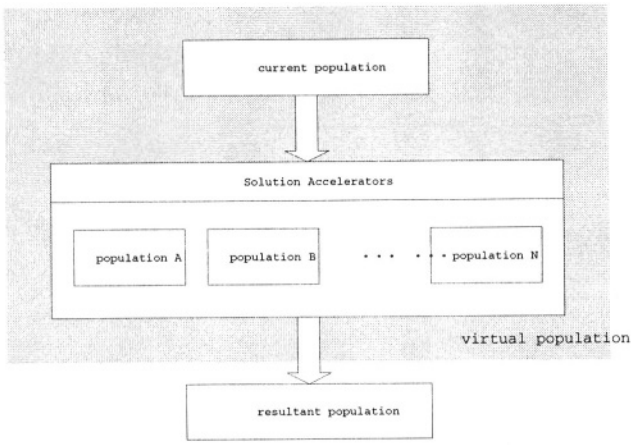


Figure 13.1. Formation of virtual and resultant populations of candidate solutions

While Fig. 13.1 shows a scheme for forming the new populations *A* to *N* through the use of solution acceleration techniques, there can be other possible ways to establish the new populations. An example of this

is to form population B by applying a solution acceleration technique to the combined current population and population A.

3. Solution Acceleration Techniques

3.1 Numerical Solution Acceleration

To implement the scheme shown in Fig. 13.1 or its variants, efficient solution acceleration techniques must first be developed. Some solution acceleration techniques have previously been employed in conventional iterative methods for determining the solutions of unknown variables in a set of simultaneous equations. In the power system field, an acceleration method was designed and embedded in the iterative Gauss-Siedel method (Grainger and Stevenson, 1994) for solving the load flow problem. In that method, the accelerated solution of variable x at the pth iteration is estimated from its solutions at the pth and the $(p-1)$th iterations according to

$$x_{acc}^{p} = x^{p-1} + \alpha(x^{p} - x^{p-1}) \tag{13.1}$$

where α is a constant coefficient. When α is unity, there is no solution acceleration. The acceleration mechanism is initiated by setting the value of α to a value greater than unity. For most electric power networks, α has been found experimentally to be about 1.6. A similar method has been designed for the iterative Newton-Raphson method to find the solutions for ill-conditioned load flow equations. In this case, the value of α is found by solving a cubic equation (Iwamoto and Tamura, 1981).

Based on the solution acceleration concept in the above technique, the authors have developed a solution acceleration methodology (Wong and Li, 1997) for improving the convergence characteristics of genetic algorithms. In that method, it is assumed that the best solution $\mathbf{V}_{best}$ in the current population P is the closest solution to the global optimum. If the assumption holds, then searching the solution space in the neighbourhood of the best candidate solution will produce solutions closer to the global optimum. This can be implemented by updating the other candidate solutions in population P according to the expression below.

$$\mathbf{V}_{acc} = \mathbf{V}_{best} + \lambda(\mathbf{V}_{best} - \mathbf{V}) \tag{13.2}$$

where $\mathbf{V}$ is a candidate solution in the current population and $\mathbf{V}_{acc}$ is the accelerated $\mathbf{V}$. Parameter λ is the acceleration factor, its value is in the range of 0 to 1. Fig. 13.2 gives a graphical interpretation to this

solution acceleration method.

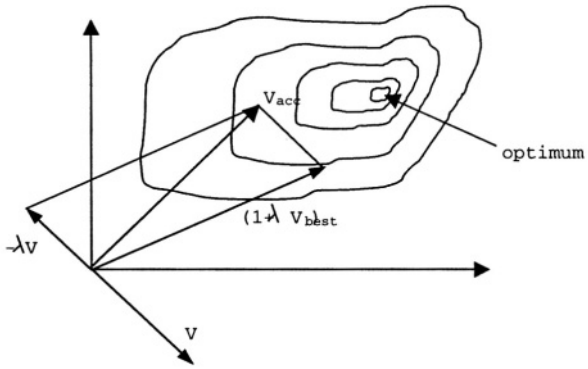


Figure 13.2. Graphical interpretation of the numerical solution acceleration method

3.2 Analytical Solution Acceleration

While the acceleration method developed in the last section is numerical, analytical acceleration techniques based on gradients can also be established. One example of the analytical acceleration techniques is to use the derivatives of the objective function to be optimised as a solution accelerator. Consider the following unconstrained optimisation problem:

$$\min \quad f(v_1, v_2, ..., v_N) \tag{13.3}$$

The accelerated candidate solution, $\mathbf{V}_{acc}$, is formed by accelerating the old candidate solution $\mathbf{V}$ using its gradient according to

$$\mathbf{V}_{acc} = \mathbf{V} + \mu\mathbf{U}[f(\mathbf{V}_{best}) - f(\mathbf{V})] \tag{13.4}$$

where $\mathbf{V} = [v_1, v_2, ..., v_N]^t$, $\mathbf{U} = [1, 1, ..., 1]^t$, $\mathbf{V}_{best}$ is the best candidate solution in the current population, and μ is a matrix of derivatives given in the following:

$$\mu = \begin{pmatrix} \left(\frac{\partial f}{\partial v_1}\right)^{-1} & 0 & \cdots & 0 \\ 0 & \left(\frac{\partial f}{\partial v_2}\right)^{-1} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \left(\frac{\partial f}{\partial v_N}\right)^{-1} \end{pmatrix}$$

The matrix μ will be the inverse of a Jacobian matrix when the problem described by equation (13.3) consists of N independent equations with N variables.

4. Accelerated GA and Acceleration Schemes

The utilisation of the different acceleration methods such as those described in Section 3 to form the new populations A to N in the virtual population will provide more diversity of the chromosomes that the evolutionary process requires for determining the global optimal solution. By incorporating the virtual population scheme in Fig. 13.1 and the solution acceleration techniques into the standard GA algorithm, an accelerated GA (AGA) can be formed. In AGA, the 2-point crossover and uniform mutation methods are used. The probability of crossover is 0.9 and that of mutation is 0.01. The Roulette Wheel selection scheme is adopted.

The following four schemes are designed and used to form the resultant population from the current population using the concept of virtual population and the developed solution acceleration techniques:

Scheme (a)
The current population is accelerated using the numerical acceleration method in Section 3.1. This scheme is denoted by the symbol $(C \Rightarrow N)$;

Scheme (b)
The current population is accelerated using the analytical acceleration method in Section 3.2. This scheme is denoted by the symbol $(C \Rightarrow A)$;

Scheme (c)
The population obtained in scheme (**a**) is further accelerated using the analytical acceleration method as in scheme (**b**) to form the resultant population. This scheme is denoted by the symbol $(C \Rightarrow N \Rightarrow A)$;

Scheme (d)
The candidate solutions in the populations found in schemes (**a**) and (**b**) are ranked according to their fitness. The best k solutions are selected to form the resultant population of size k. This scheme is denoted by the symbol $(C \Rightarrow N \cup C \Rightarrow A)$.

In the above schemes, scheme (**a**) which employs only the numerical acceleration has resently been reported in (Wong and Li, 1997). It represents the most recent accelerated genetic algorithm and has been

applied to the load flow problem of practical power systems (Wong et al., 1997) including the IEEE 118-node power system. This scheme has also been incorporated into EP for solving the environmentally-constrained economic dispatch problem in the operation of power systems (Wong and Yuryevich, 1998). The new schemes (**b**)-(**d**) will be compared with scheme (**a**) in the present application studies.

5. Validation of Methods

The AGA was applied to solving the Branin RCOS, De Jong 1 and Colville problems (Michalewicz, 1996). In solving these problems, the four schemes described in Section 4 were used. AGA was executed on a Pentium Pro 200 computer for a total of 100 trials with each of the four schemes. In each trial, the maximum number of generations is 500. For all trials, the probability of crossover is set to 0.9 and the probability of mutation is set to 0.01.

Table 13.1. Performance Evaluation

Scheme	Branin RCOS			De Jong 1			Colville		
	S_{min}	$\bar{g}$	E	S_{min}	$\bar{g}$	E	S_{min}	$\bar{g}$	E
a	120	2.00	240.00	30	18.09	542.70	-	-	-
b	30	5.06	151.80	26	6.28	163.28	200	249.94	49988
c	20	3.00	60.00	22	4.00	88.00	200	30.19	6038
d	30	3.20	96.00	30	5.92	177.60	200	31.34	6268

S_{min} : minimum population size
$\bar{g}$: average number of executed generations per trial
E : number of evaluations = $S_{min} * \bar{g}$

The results at the minimum population size found for each scheme are tabulated in Table 13.1. The definition of minimum population size has been given in Section 1. The associated average execution times in Table 13.2 are over 100 trials. The results in Table 13.1 show that the earlier acceleration method (Wong and Li, 1997), scheme (**a**), cannot solve the Colville problem. While it can solve the Branin RCOS and De Jong 1 problems, it requires a bigger minimum population size S_{min} and a larger number of evaluations E than the new schemes (**b**) - (**d**).

Table 13.2. Average execution time (second) per trial

Scheme	Branin RCOS	De Jong 1	Colville
a	0.004210	0.013730	-
b	0.004110	0.004510	1.271270
c	0.002400	0.002900	0.163540
d	0.009710	0.020020	3.397990

Amongst all the new schemes, scheme (**c**) is the best because it requires (i) the least minimum population size and much lower number of evaluations for solving the Branin RCOS and De Jong 1 problems and (ii) the lowest number of evaluations for solving the Colville problem. While the number of evaluations can be a good index for indicating the performance of the schemes, the average executing time per trial is another important index. Scheme (**c**) requires the least computing time. For the Colville problem, it takes only 4.8 % of the computing time needed by scheme (**d**) and only 12.9 % of that of scheme (**b**). The behaviour of schemes (**c**) can be examined further by applying it to problems described by sets of simultaneous equations instead of a single function with multiple variables as in the present studies. For completeness, the value of λ in equation 13.2 employed in schemes (**a**), (**c**) and (**d**) for the three problems are tabulated in Table 13.3.

Table 13.3. Value of acceleration coefficient λ

Scheme	Branin RCOS	De Jong 1	Colville
a	0.2	0.3	-
b	-	-	-
c	0.2	0.3	1.0
d	0.2	0.3	1.0

The standard GA was also applied to solving the three test problems. However, it cannot solve the Colville problem for a population size of 200 although it can deal with the Branin RCOS and De Jong 1 problems with success rates of 29% and 52% respectively. The average number of generations needed by standard GA for these two cases are 404 and 394 respectively. The standard GA is extremely inferior to AGA with any of the acceleration schemes (**a**)-(**d**).

6. Further Improvement: Refined Scheme (c)

While scheme (c) has been found above to be the best scheme amongst schemes (a) - (d), its computational speed can be further reduced by refining it using the following procedure:

Instead of accelerating all the chromosomes in the population obtained in scheme (a) using the analytical acceleration method, only a percentage of the best chromosomes is accelerated. While this retains the power of the analytical acceleration method, this also reduces the computing time. The accelerated chromosomes and the remaining chromosomes in the original population combine to form the resultant population. For

convenience, this procedure is referred to as Refined Scheme (c). Applying Refine Scheme (c) to the Branin RCOS and De Jong F1 problems, it has been found experimentally that only between 20% and 40% of the chromosomes need to be accelerated analytically and the average execution time per trial is 0.001673 sec and 0.002438 sec. respectively. Comparing the execution times in row 4 of Table 13.2, it can be observed that there is a large execution time reduction in the Branin RCOS case and some execution time reduction in the De Jong F1 case. For the Colville problem, the range of percentage found is from 30% to 60% and the average execution time per trial is 0.034588 sec. Comparing to 0.16354 sec. as shown in Table 13.2 for the 100% case, that is, the original scheme (c), there is a very large reduction in execution time. These results show the effectiveness of the Refine Scheme (c).

7. The Load Flow Problem in Electrical Power Networks

The load flow or power flow problem can be described mathematically as follows. Consider there are a total number of N nodes in a power system. At any node i, the nodal active power, P_i , and reactive power, Q_i, are given by:

$$P_i = E_i \sum_{j=1}^{N} (G_{ij}E_j - B_{ij}F_j) + F_i \sum_{j=1}^{N} (G_{ij}F_j + B_{ij}E_j) \qquad (13.5)$$

$$Q_i = F_i \sum_{j=1}^{N} (G_{ij}E_j - B_{ij}F_j) - E_i \sum_{j=1}^{N} (G_{ij}F_j + B_{ij}E_j) \qquad (13.6)$$

where G_{ij} and B_{ij} are the $(i,j)th$ element of the admittance matrix. E_i and F_i are real and imaginary parts of the voltage at node i. If node i is a PQ-node where the load demand is specified, then the mismatches in active and reactive powers, ΔP_i and ΔQ_i respectively, are given by

$$\Delta P_i = |P_i^{sp} - P_i| \qquad (13.7)$$

$$\Delta Q_i = |Q_i^{sp} - Q_i| \qquad (13.8)$$

in which P_i^{sp} and Q_i^{sp} are the specified active and reactive powers at node i. When node i is a PV-node, the magnitude of the voltage, V_i^{sp} and the active power generation at i are specified. The mismatch in voltage magnitude at node i can be defined as

$$\Delta V_i = |V_i^{sp} - V_i| \tag{13.9}$$

and the active power mismatch is given in eqn. (13.7). In eqn. (13.9), V_i is the calculated nodal voltage at PV-node i and is given by

$$V_i = \sqrt{E_i^2 + F_i^2} \tag{13.10}$$

The unknown variables in this problem are (i) the voltages at the PQ-nodes and (ii) the real and imaginary parts of the voltages at the PV-nodes and the reactive powers of the generators connected to the PV-nodes. It is required to determine the values of the variables such that the mismatches in eqns. (13.7) - (13.9) are zero.

Apart from solving the load-flow problem by the conventional methods, the problem can be viewed as an optimisation problem, in which an objective function H is to be minimised. The objective function can be defined as the total mismatch and be expressed as

$$H = \sum_{k \in N_{pq}+N_{pv}} |P_i^{sp} - P_i|^2 + \sum_{k \in N_{pq}} |Q_i^{sp} - Q_i|^2 + \sum_{k \in N_{pv}} |V_i^{sp} - V_i|^2 \tag{13.11}$$

where N_{pq} , N_{pv} are the total numbers of PQ-nodes and PV-nodes respectively. When the power flow problem is solvable, the value of H is zero or in the vicinity of zero at the end of the optimisation process. If the problem is unsolvable, the value of H will be greater than zero. The square root of H can be regarded as the minimum distance between the solution point in the unsolvable region and the boundary of the solvable region.

8. Accelerated Constrained Genetic Algorithms for Load Flow Calculation

A constrained-GA load flow (CAGLF) algorithm (Wong et al., 1997) has been developed by the authors for minimising the objective function in eqn. 13.11 for the load flow problem. The pseudo code of the algorithm is described in Table 13.4 below.

The details of the constraint satisfaction technique in the CGALF algorithm can be found in Wong et al. (1997). Central to the constraint satisfaction technique is to the modification of the network nodal voltages in the chromosomes according to the constraint voltage expressions

which are derivable from eqns. 13.5 - 13.10 and by setting the mismatches in eqns. 13.7 - 13.9 to zero. To improve the robustness and computational speed of CGALF, the Refine Scheme (c) has been incorporated into CGALF to form the CGALFB algorithm. In CGALFB only 25% of the chromosomes in the population are accelerated analytically. For comparison purposes, scheme (a) is also incorporated in CGALF to form the CGALFA algorithm. The performances of CGALFA and CGALFB are illustrated in the next section by their applications to the Klos-Kerner 11-node power system (Klos and Kerner, 1975).

9. Klos-Kerner 11-Node System Studies

Fig, 13.3 shows the Klos-Kerner 11-node (KK11) test system. Node 1 is the slack node at a voltage level of 1.05 p.u. and nodes 5 and 9 are PV nodes with target voltages of 1.05 p.u. and 1.0375 p.u. respectively. The network data of KK11 are given in Table 13.5. The system loading condition is very heavy and is given in Table 13.6.

Under the given heavy-load condition, in Reference (Klos and Kerner, 1975), Klos and Kerner presented two very near normal solutions, the nodal voltage solutions of which only differ in the third or fourth decimal place. However, the total squared mismatch of the two solutions are 0.058745 and 0.058730. If 0.001 p.u. on 100 MVA base is assumed to be the solution tolerance, the active and reactive power mismatches at the nodes are greater than this tolerance. These power mismatches are therefore too high and the two near normal solutions found by Klos and Kerner under the specified loading condition may not be valid. This test case has been presented to a Newton-Raphson Load Flow (NRLF) program. Using 0.001 p.u. on 100 MVA base as the tolerance for the power mismatch and 0.001 p.u. voltage as the tolerance for voltage mis-

Table 13.4. Pseudo code of the CGALF algorithm Set the generation

```
Set the generation counter g to 0;
Initialise population of chromosomes P(g) at generation g;
Evaluate fitness of chromosomes in P(g);
While notterminate {
   Generate P(g + 1) from P(g);
   Update P(g + 1) using the constraint satisfaction technique;
   Accelerate P(g + 1) using acceleration techniques g = g + 1;
   Evaluate fitness of chromosomes of P(g);
   Store fittest chromosome;
  }
```

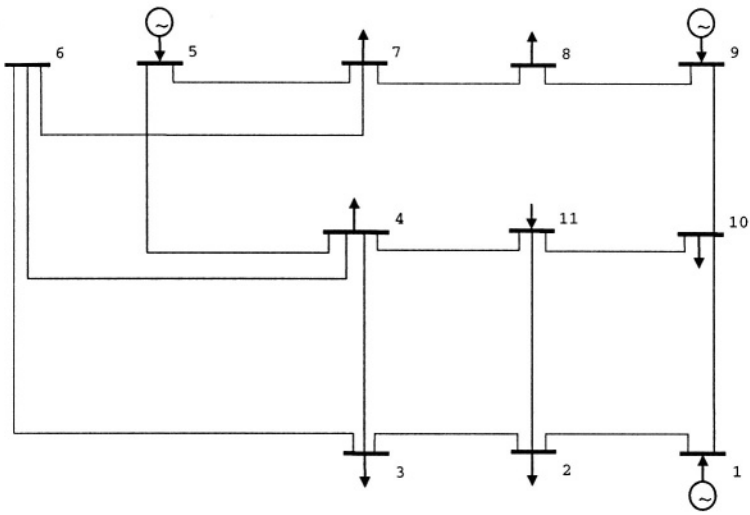


Figure 13.3. KK11 test system

Table 13.5. Network data of the KK 11 test system

Node Connection (*sending*)	(*receiving*)	Series Impedance (*p.u.*)		Shunt Admittance (*p.u.*)
1	2	0.00250	0.0188	0.640
1	10	0.00500	0.0375	1.280
2	3	0.00125	0.0125	0.258
2	11	0.00625	0.0625	1.600
3	4	0.00375	0.0312	0.640
3	6	0.00438	0.0375	1.280
4	5	0.00312	0.0312	0.960
4	6	0.00312	0.0312	0.960
4	11	0.00188	0.0125	0.320
5	7	0.00625	0.0625	1.600
6	7	0.00625	0.0625	1.600
7	8	0.00125	0.0125	0.256
8	9	0.00188	0.0188	0.480
9	10	0.00125	0.0125	0.320
10	11	0.00312	0.0312	0.960

system power base is 100 MVA

match, NRLF fails to converge to a solution after 20 iterations and the total squared mismatch H is 0.005687. Corresponding to the adopted tolerance of 0.001 p.u., the value of H should be equal to or smaller than 0.000020. One possibility here is that the given loading condition leads to the situation that the load-flow equation set is unsolvable. The second possibility is that the given loading condition is at the steady-state ceiling point of the system, that is at the saddle-nose bifurcation point, at which the Jacobian matrix in NRLF is singular. The third possibil-

Table 13.6. Specified generation, loads and nodal voltages for KK 11 test system

Node No. (i)	V_i^{sp} ($p.u.$)	θ_i^{sp} ($deg.$)	P_i^{sp} ($p.u.$)	Q_i^{sp} ($p.u.$)
1	1.05	0.0	-	-
2	-	-	-9.00	-2.00
3	-	-	-16.5095	-1.00
4	-	-	-25.00	-2.00
5	1.05	-	10.00	-
6	-	-	0.00	0.00
7	-	-	-12.00	-0.80
8	-	-	-4.00	-0.50
9	1.0375	-	25.00	-
10	-	-	-8.00	-2.00
11	-	-	9.00	6.30

ity is that this loading condition is near the saddle-nose point and the Jacobian is nearly singular causing NRLF to diverge.

9.1 Results Obtained by CGALFA

The CGALFA algorithm developed has been applied to the system under the specified heavy-load condition. The parameter settings for executing CGALFA are as follows: The probability of crossover is 0.9, the probability of mutation is 0.01, the population size is 100, and the maximum number of generations or iterations is 100. The initial candidate solution chromosomes are formed randomly in the voltage range from 0.0 p.u. to 1.2 p.u. and in the voltage phase angles range between 0° and –180°. These ranges are also employed in the mutation operation.

Table 13.7. Best solution for the heavy-load case by CGALFA

Node No. (i)	V_i ($p.u.$)	θ_i ($deg.$)	Mismatches ΔP_i ($p.u.$)	ΔQ_i ($p.u.$)	ΔQ_i ($p.u.$)	H
1	1.050000	0.000000	-	-	-0.000131	
2	0.722943	-36.733223	-0.001387	-0.004189	-	
3	0.712220	-57.940029	-0.005795	-0.002203	-	
4	0.816123	-56.339577	-0.003992	-0.002477	-	
5	1.050146	-40.259895	-0.004489	-	-0.000146	
6	0.816222	-54.381390	-0.003340	-0.002306	-	0.000131
7	0.904489	-45.357384	-0.002360	-0.000367	-	
8	0.920475	-34.486439	-0.001586	0.000821	-	
9	1.037502	-15.482958	-0.000944	-	-0.000002	
10	0.964997	-20.607037	-0.000549	-0.000544	-	
11	0.887599	-39.122875	-0.001139	-0.001696	-	

The solution found by CGALFA is tabulated in Table 13.7. The total squared mismatch is 0.000131 due to the voltage magnitude mismatch at node 5 and the active and reactive power mismatches at the PQ nodes. The fact that the voltage magnitude mismatch at node 5 is non-zero indicates that the present load flow problem is unsolvable. Owing to the insolvability of the problem, NRLF does not converge to a solution. It follows that the two very near solutions given by Klos and Kerner (1975) are not valid solutions. By reducing the original system load demand by 0.04% and by sharing the reduction evenly throughout all the PQ nodes in the system, a ceiling point is found and the solution is unique subject to the 0.001 p.u. tolerance. This solution is summarised in Table 13.8. It is obtained at the 12th iteration where the total squared mismatch is 0.0000048.

Table 13.8. Best solution for the heavy-load case by CGALFA with 0.04% load reduction

Node No.	V_i	θ_i	Mismatches			H
			$\triangle P_i$	$\triangle Q_i$	$\triangle Q_i$	
(i)	$(p.u.)$	$(deg.)$	$(p.u.)$	$(p.u.)$	$(p.u.)$	
1	1.050000	0.000000	-	-	-	
2	0.724861	-36.629562	-0.000299	-0.000759	-	
3	0.714262	-57.727352	-0.000879	-0.000838	-	
4	0.817576	-56.150093	-0.000944	-0.000234	-	
5	1.050000	-40.104637	-0.000324	-	0.000000	
6	0.817744	-54.203972	-0.000633	-0.000602	-	0.00000480
7	0.905111	-45.222538	-0.000387	0.000029	-	
8	0.920962	-34.376129	-0.000500	-0.000191	-	
9	1.037500	-15.398665	-0.000324	-	0.000000	
10	0.965454	-20.534266	-0.000017	-0.000024	-	
11	0.888948	-38.999729	-0.000476	-0.000302	-	

The execution time is about 4.58 sec. per iteration on a Pentium 90 personal computer. However, out of 30 trials, CGALFA converged to the solution only two times. Fig. 13.4 shows the convergence characteristic of CGALFA in 100 iterations for this case. In comparison, NRLF fails to converge to a solution at this revised loading condition. In addition, it fails to converge until the original total loading of the system is reduced by 0.05%.

9.2 Results Obtained by CGALFB

On applying the CGALFB algorithm to the KK11 test system under the heavy-load condition but with the total system load demand reduced by 0.04% as in the above case, the parameter settings for executing CGALFB are the same as those for CGALFA in Section 9.1 above but

with a population size of 50 which is half of the previous size. With voltage magnitude range of 0.9 to 1.2 p.u. and phase range of 0.0 to -30.0 deg. for the initialisation of the population and for mutation. CGALFB took only an average of 9 iterations to converge and the results are tabulated in Table 13.9. The computing time on a Pentium Pro 200 computer was 0.23 seconds per iteration on average. Out of 100 trials, all trials converged to the solution. This study confirms that the CGALFB algorithm is far superior to CGALFA in the quality of the solution and convergence characteristics.

At 0.05% or more system load reduction, CGALFB takes 3 iterations to converge to the solution when the population size is 50 and it takes only 5 to 6 iterations with a population size of 4, which is also found to be the minimum size. This performance cannot be achieved by CGALF and CGALFA.

10. Conclusions

The concept of virtual population and numerical and analytical solution acceleration techniques for improving the robustness of genetic algorithms have been developed. This paper has also introduced a definition of the minimum population size. Four acceleration schemes plus Refined Scheme (c) have been proposed and incorporated into GA to

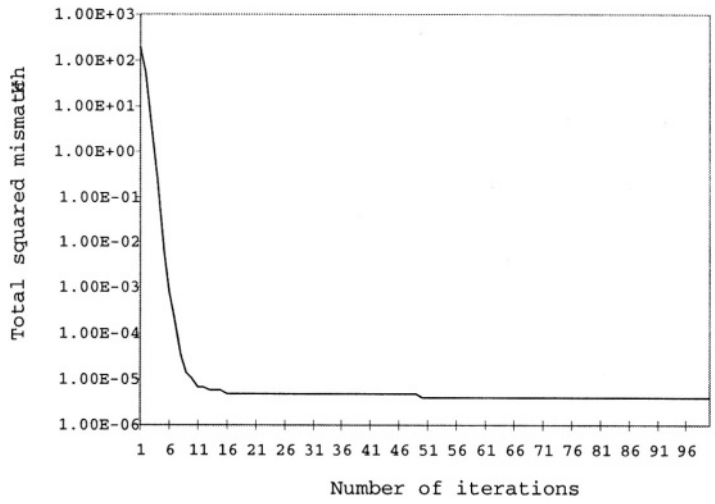


Figure 13.4. Convergence characteristic of CGALFA

Table 13.9. Solution for the heavy-load case by CGALFB with 0.04% load reduction

Node No. (i)	V_i ($p.u.$)	θ_i ($deg.$)	Mismatches $\triangle P_i$ ($p.u.$)	Mismatches $\triangle Q_i$ ($p.u.$)	Mismatches $\triangle Q_i$ ($p.u.$)	H
1	1.050000	0.000000	-	-	-	
2	0.724098	-36.665249	-0.000322	-0.000970	-	
3	0.713425	-57.805126	-0.000910	-0.000498	-	
4	0.816968	-56.216946	-0.000984	-0.000608	-	
5	1.050000	-40.158127	-0.000500	-	0.000000	
6	0.817096	-54.267612	-0.000500	-0.000497	-	0.00000736
7	0.904844	-45.266586	-0.000594	0.000347	-	
8	0.920765	-34.410599	-0.000819	-0.000348	-	
9	1.037500	-15.423858	-0.000498	-	0.000000	
10	0.965285	-20.555542	-0.000774	-0.000496	-	
11	0.888394	-39.040169	-0.000815	-0.000360	-	

form the AGA. AGA has been validated by applying it to the Branin RCOS, De Jong F1 and Colville problems. The results have shown that the acceleration scheme Refine Scheme (c) has the best performance. This finding is applicable to other evolutionary optimisation algorithms.

By incorporating scheme (a) and Refine Scheme (c) in the CGALF algorithm, CGALFA and CGALFB have been formed respectively. Using CGALFA, it has also been found that the solvability of the load flow problem of KK11 under heavy-load condition can be achieved by a reduction in load demand of 0.04%. CGALFA offers much better performance than the Newton Raphson method as the latter method fails to converge at the vicinity of the ceiling load point. The same application studies have also been carried out using CGALF and CGALFB. The results have shown that CGALF is inferior to CGALFA. They have also shown that CGALFB is far superior to CGALFA in robustness and convergence characteristics. This finding confirms the effectiveness and power of the virtual population concept and acceleration methods, in particular Refine Scheme (c) in Section 6, when applied to CGALF for solving the load flow problem. CGALFB has also been applied to determining the network loadability limit of some IEEE standard power systems and the results can be found in Reference (Wong et al., 1999).

References

Goldberg, D. (1989). *Genetic Algorithms in Search, Optimisation and Machine Learning.* Addison-Wesley.

Grainger, J. and Stevenson, J. (1994). *Power System Analysis.* McGraw-Hill, Inc.

Holland, J. (1975). *Adaption in Natural and Artificial Systems.* Ann Arbor: University of Michigan Press.

Iwamoto, S. and Tamura, Y. (1981). A load flow calculation method for ill-conditioned power systems. *IEEE Trans. on Power Apparatus and Systems,* PAS-100(4):1736–1743.

Klos, A. and Kerner, A. (July 1975). The non-uniqueness of load flow solutions. *Proc. 5th Power Sys-tem Computation Conf. (PSCC), Cambridge, UK,* V.3.1:1–8.

Michalewicz, Z. (1996). *Genetic algorithms + data structures = evolution programs, 3rd rev. extended ed.* Springer-Verlag.

Wong, K. and Li, A. (1997). A technique for improving the convergence characteristic of genetic algorithms and its application to a genetic-based load flow algorithm. *Simulated Evolution and Learning, JH Kim, X. Yao, T. Furuhasi (Eds), Lecture Notes in Artificial Intelligence 1285,* 167–176.

Wong, K., Li, A., and Law, M. Y. (1997). Development of constrained genetic-algorithm load-flow method. *IEE Proc.-Gener. Transm. Distrib.,* 144(2):91–99.

Wong, K., Li, A., and Law, M. Y. (1999). Advanced constrained-genetic-algorithm load flow method. *IEE Proc.-Gener. Transm. Distrib.,* 146(6): 609–616.

Wong, K. and Yuryevich, J. (1998). Evolutionary-programming-based algorithm for environmentally-constrained economic dispatch. *IEEE Trans. on Power Systems,* 13(2):301–306.

This page intentionally left blank

VII

APPLICATION OF EAS TO THEORETICAL PROBLEMS

This page intentionally left blank

Chapter 14

METHODS FOR THE ANALYSIS OF EVOLUTIONARY ALGORITHMS ON PSEUDO-BOOLEAN FUNCTIONS*

Ingo Wegener

Abstract Many experiments have shown that evolutionary algorithms are useful randomized search heuristics for optimization problems. In order to learn more about the reasons for their efficiency and in order to obtain proven results on evolutionary algorithms it is necessary to develop a theory of evolutionary algorithms. Such a theory is still in its infancy. A major part of a theory is the analysis of different variants of evolutionary algorithms on selected functions. Several results of this kind have been obtained during the last years. Here important analytical tools are presented, discussed, and applied to well-chosen example functions.

1. Introduction

Evolutionary algorithms are randomized search heuristics with many applications, e.g., in optimization, adaptation, classification, control systems, or learning. Here we focus on optimization (for an overview on the whole area we refer to Bäck, Fogel, and Michalewicz, 1997; Fogel, 1995; Goldberg, 1989; Holland, 1975; Schwefel, 1995). Despite the many successful experiments with evolutionary algorithms a theory on evolutionary algorithms is still in its infancy. This holds in particular if one compares the state of the art with the situation on problem-specific deter-

*This work was supported by the Deutsche Forschungsgemeinschaft (DFG) as part of the Collaborative Research Center "Computational Intelligence" (SFB 531).

ministic exact optimization algorithms (Cormen, Leiserson, and Rivest, 1990), deterministic approximation algorithms (Hochbaum, 1997), or randomized optimization and approximation algorithms (Motwani and Raghavan, 1995). One reason is that evolutionary algorithms have been developed by engineers, while the other disciplines have been created by theoreticians (leading sometimes to a lack of experimental knowledge). Moreover, the fundamental idea of evolutionary algorithms is to obtain robust problem-independent search heuristics with a good behavior on many problems from a large variety of problems (this statement remains true, although many evolutionary algorithms also use problem-specific components). This variety of problems makes the analysis of evolutionary algorithms much harder than the analysis of problem-specific algorithms (which often are designed in order to make an analysis possible). Nevertheless, progress on the design and the application of evolutionary algorithms will gain a lot from a theoretical foundation. Nowadays, we are able to analyze evolutionary algorithms without crossover on many functions and evolutionary algorithms with crossover on some functions. The functions are not examples from real-world applications but example functions describing some typical issues of functions (fitness landscapes) or are chosen to show some extreme behavior of evolutionary algorithms. Also very strange functions can be useful in order to disprove widely accepted conjectures or to show the differences between variants of evolutionary algorithms. Altogether, we find a list of interesting theoretical results on evolutionary algorithms, in particular, during the last years. The purpose of this contribution is to present some of the most important tools for such results.

In Section 2, we discuss differences between discrete and non-discrete state spaces and why we investigate the optimization of pseudo-boolean functions $f: \{0,1\}^n \to \mathbb{R}$. The aim of an analysis of an evolutionary algorithm is the investigation of certain performance measures. This paper focusses on expected run times and the success probability within reasonable time bounds. The reasons for this decision are presented in Section 3. Since several example functions are used for different purposes all these functions are defined in Section 4. The following three sections show how tail inequalities (Section 5), the coupon collector's theorem (Section 6), and results on the gambler's ruin problem (Section 7) can be applied to the analysis of evolutionary algorithms. Another main idea is to measure the progress of an evolutionary algorithm not with respect to the considered fitness function but with some cruder scale. In Section 8 and Section 9, upper and lower bounds on the expected run time of evolutionary algorithms are proved using levels based on intervals of fitness values. The method of using so-called potential functions for the

analysis of algorithms is well established. We discuss the first successful application of this powerful tool for evolutionary algorithm in Section 10. Finally, in Section 11, we present the method of designing "typical runs" of an evolutionary algorithm. Then we can bound the time of a typical run and can estimate the failure probability, i.e., the probability that a run is not typical.

2. Optimization of pseudo-boolean functions

Although many of our methods and results can be transferred to the non-discrete case, we focus here on the somehow simpler or clearer case of discrete functions. We also abstract from possible constraints which leads to the search space $S = \{0,1\}^n$.

Definition 4. Functions $f: \{0,1\}^n \to \mathbb{R}$ are called pseudo-boolean. Without loss of generality we investigate the maximization of pseudo-boolean fitness functions. While technical systems lead to non-discrete search spaces, the whole area of combinatorial optimization leads to our scenario. Pseudo-boolean functions always have optimal search points. Search heuristics evaluate each point of the search space in expected finite time implying that expected run times are finite. Moreover, the search space has some nice features. The minimal Hamming distance between different points is 1 while the maximal distance is n. Hence, evolutionary algorithms with fixed mutation probabilities or fixed expected length of mutation steps can optimize pseudo-boolean functions efficiently. For fitness functions on $\mathbb{R}^n$ and even on some compact subspace of $\mathbb{R}^n$ it is necessary to decrease the expected length of steps in order to approximate the optimum. Nevertheless, the chance of meeting the optimum exactly is often 0.

In order to present the techniques to analyze evolutionary algorithms for simple examples we often investigate the perhaps simplest randomized search heuristic belonging to the class of evolutionary algorithms.

Algorithm 1 (1+1)EA

- *Choose* $x = (x_1, \ldots, x_n) \in \{0,1\}^n$ *randomly.*
- *Let* x' *be the result of mutating* x*, i.e., the bits of* x'_i *are generated independently and* $x'_i = \overline{x}_i$ *with the mutation probability* $1/n$ *and* $x'_i = x_i$ *otherwise.*
- *Replace* x *with* x' *iff* $f(x') \geq f(x)$.
- *Repeat the last two steps until a stopping criterion is fulfilled.*

Often we consider the Markov process describing the randomized search with the (1+1)EA as infinite process without stopping rule. Then we are interested in the first point of time where something nice happens, e.g., the first point of time where a global optimal search point is evaluated. The investigation of the (1+1)EA is less limited as one may believe. We may use the multi-start option, i.e., we consider p independent runs of the (1+1)EA. This is often better or at least as good as the consideration of a population of size p. The independency of runs ensures the diversity. For larger populations, one has to ensure the diversity by certain tricks. A population size of 1 does not admit crossover. However, the analysis of evolutionary algorithms without crossover is difficult enough and we consider evolutionary algorithms with crossover only in Section 11. The mutation probability of $1/n$ is the most often recommended choice (Bäck, 1993). Alternating mutation probabilities by self-adaptation have been investigated by Beyer (1996) and Bäck (1998) and a dynamic variant of the (1+1)EA has been analyzed by Jansen and Wegener (2000a).

3. Performance measures

The analysis of an algorithm is the task to describe quantitatively the behavior of the algorithm. Let X_f be the random variable measuring the first point of time when some event happens. The most natural choice of such an event is that an optimal search point is evaluated. However, one may also consider the case that some search point whose fitness is sufficiently close to the fitness of an optimal point or the case that some search point is sufficiently close to an optimal point. The run time analysis of a randomized algorithm consists of

- the computation or estimation of $E(X_f)$,
- the analysis of the so-called success probability distribution $Pr(X_f \leq t)$, and
- the analysis of the best case, worst case, or average case of $E(X_f)$ and $Pr(X_f \leq t)$ with respect to functions f from some class F of functions.

The investigation of the success probability includes the investigation of multi-start variants. E.g., if $Pr(X_f \leq t) \geq \frac{1}{n}$, n independent runs have after t steps a success probability of at least $1 - (1 - \frac{1}{n})^n \geq 1 - e^{-1}$, $n \log n$ independent runs improve the success probability to $1 - O(\frac{1}{n})$ and n^2 runs even to $1 - 2^{-\Omega(n)}$. There are examples where $E(X_f)$ grows exponentially while the success probability after a polynomial number of steps is bounded below by a positive constant.

Expected run time and success probability are global performance measures and are those performance measures which typically are used for randomized algorithms (see Motwani and Raghavan, 1995). In the theory of evolutionary algorithms many other aspects are considered. These other performance measures are of certain value, but we think that their analysis finally is only a tool to get results on the global behavior. In order to understand the global behavior it is useful to understand the local behavior. Quality gain and progress rate (see Bäck, Fogel, and Michalewicz, 1997) are such local performance measures which describe the behavior of a single step. The schema theorem also is a result which guarantees a certain behavior for one step. For Markov processes (like evolutionary algorithms) the transition probabilities for one step determine the global behavior. However, the local performance measures are so-called "insufficient statistics" implying that in general it is not possible to deduce statements on the global behavior. Examples where these local performance measures give "wrong hints" are contained in Jansen and Wegener (2000b). Our global performance measures describe the behavior within reasonable time bounds. We think that the limit behavior is of much less interest. Even for state spaces like $S = \mathbb{R}^n$ we look for a good behavior within reasonable time bounds and not in the limit. Finally, interesting results have been obtained by modelling evolutionary algorithms as dynamical systems. However, this model implicitly works with infinite populations and one has to carefully investigate the difference between infinite populations, finite populations, and populations of reasonable size. Rabani, Rabinovich, and Sinclair (1998) have obtained first results how to control the difference between these cases.

4. Selected functions

Here we give an overview on the functions investigated in this paper as examples.

Definition 5. A pseudo-boolean function $f: \{0,1\}^n \to \mathbb{R}$ is a degree-k function with N non-vanishing terms if it can be represented as

$$f(x_1, \ldots, x_n) = \sum_{1 \le i \le N} w_i \prod_{j \in S_i} x_j$$

where $w_i \in \mathbb{R} - \{0\}$ and the size of the sets $S_i \subseteq \{1, \ldots, n\}$ is bounded above by k. Degree-1 functions are called linear and degree-2 functions are called quadratic.

The following two linear functions are of particular interest. They are the extreme examples of equal and strongly different weights.

Definition 6.

$$\text{ONEMAX}(x_1, \ldots, x_n) = ||x|| := x_1 + \cdots + x_n.$$
$$\text{BV}(x_1, \ldots, x_n) = 2^{n-1}x_1 + 2^{n-2}x_2 + \cdots + 2x_{n-1} + x_n,$$
where BV stands for binary value.

Unimodal functions are those functions where the global optimum is unique and can be reached from each point by 1-bit mutations.

Definition 7. A pseudo-boolean function is called unimodal if it has a unique global optimum and all other search points have a Hamming neighbor with a larger fitness.

We consider the special unimodal function LO (leading ones) and the class of path functions.

Definition 8.
$\text{LO}(x_1, \ldots, x_n) = \max\{i | x_1 = \cdots = x_i = 1 \text{ and } (i = n \text{ or } x_{i+1} = 0)\}$ measures the length or the longest prefix consisting of ones only.

Definition 9. A path p starting at $a \in \{0,1\}^n$ is defined by a sequence of points $p = (p_0, \ldots, p_l)$ where $p_0 = a$ and $H(p_i, p_{i+1}) = 1$. A function $f: \{0,1\}^n \to \mathbb{R}$ is a path function with respect to the path p if $f(p_{i+1}) > f(p_i)$ for $0 \leq i \leq l-1$ and $f(b) < f(a)$ for all b outside the path.

Definition 10. SP (short path) is defined with respect to the path $p = (p_0, \ldots, p_n)$ where $p_i = 1^i 0^{n-i}$ by

$$\text{SP}(x_1, \ldots, x_n) = \begin{cases} n + i & \text{if } x = p_i \\ n - ||x|| & \text{otherwise.} \end{cases}$$

Long path functions are defined on exponentially long paths. They were first introduced by Horn, Goldberg, and Deb (1994). Their long path functions admit the possibility of shortcuts namely a mutation flipping $O(1)$ bits and replacing p_i with p_j where $j - i$ is exponentially large. This can make them easy for evolutionary algorithms as shown by Rudolph (1997) who also introduced long path functions with the additional property that for each p_i there is at most one successor on the path with Hamming distance d if $d \leq n^{1/2}$. Moreover, he has specified the fitness values outside the path to obtain a unimodal function which is difficult for the (1 + 1)EA. The exact definition is not necessary here. We call this function LP (long path).

Another interesting issue is the investigation of evolutionary algorithms in the presence of plateaus, i.e., connected subsets of the input space with constant fitness and with only few neighbored points with a better fitness. The following function is a good example for such a function.

Definition 11. The function SPP (short path as plateau) is defined with respect to the path $p = (p_0, \ldots, p_n)$ where $p_i = 1^i 0^{n-i}$ by

$$\mathrm{SPP}(x_1, \ldots, x_n) = \begin{cases} 2n & \text{if } x = 1^n \\ n & \text{if } x = p_i, i < n \\ n - ||x|| & \text{otherwise.} \end{cases}$$

Finally, we introduce two special functions where the first one has the property of giving wrong hints and the second one is a special example for the investigation of the power of crossover.

Definition 12.

$$\mathrm{TRAP}(x_1, \ldots, x_n) = \begin{cases} n & \text{if } x = 1^n \\ n - ||x|| & \text{otherwise.} \end{cases}$$

$$\mathrm{JUMP}^m(x_1, \ldots, x_n) = \begin{cases} ||x|| & \text{if } ||x|| < n - m \text{ or } ||x|| = n \\ n - ||x|| & \text{otherwise.} \end{cases}$$

5. Tail inequalities

Tail inequalities are useful to turn expected run times into upper time bounds which hold with overwhelming probability. Moreover, for many intermediate results, expected values sometimes are useless. A mutation flips on the average one bit, but we are interested in the probability of flipping more than k bits simultaneously. A random search point contains on the average $n/2$ ones, but what do we know about the probability of less than, e.g., $n/3$ ones? The simplest tail inequality is Markov's inequality which is the basis of other tail inequalities. We present the result with its proof to show the underlying ideas.

Theorem 1 *(Markov's inequality) Let X be a random variable taking only non-negative values. Then for all* $t > 0$

$$\Pr(\mathrm{X} \geq \mathrm{t}) \leq \mathrm{E(X)/t}.$$

The result follows by estimating all X-values from [0,t) by 0 and all other X-values by t:

$$\begin{aligned} E(X) &= \sum_x x \cdot \Pr(X = x) = \sum_{x<t} x \cdot \Pr(X = x) + \sum_{x \geq t} x \cdot \Pr(X = x) \\ &\geq 0 + t \cdot \Pr(X \geq t). \end{aligned}$$

In particular, $\Pr(X \geq 2E(X)) \leq 1/2$ or $\Pr(X < 2E(X)) \geq 1/2$ proving that with a probability of at least $1 - (1/2)^k$ one out of k independent trials (or runs) has a value of less than $2E(X)$.

Other tail inequalities like Tschebyscheff's inequality and Chernoff's inequality are applications of Markov's inequality. In the first case the probability of large deviations from the expected value is bounded, i.e., $\Pr(|X - E(X)| \geq \varepsilon)$. Before applying Markov's inequality the inequality is replaced with the equivalent inequality $|X - E(X)|^2 \geq \varepsilon^2$. E.g., for $\varepsilon = 1/2$, the estimation of x^2 by 0 for $x < 1/4$ is more precise than the estimation of x by 0. For $x \geq 1/4$, x^2 is estimated by 1/16 instead of estimating x by 1/4. For small $x \geq 1/4$ this is a better estimate, but for larger x it gets worse. Hence, one may hope to get sometimes better results. Chernoff has used the "even more convex" function e^{tx} and $X \leq t$ is replaced with $e^{-sX} \geq e^{-st}$ for an appropriate value of s. We summarize the results (for the proofs see Motwani and Raghavan (1995)).

Theorem 2 *(Tschebyscheff's inequality) Let X be a random variable whose variance V(X) exists. Then for all* $\varepsilon > 0$

$$\Pr(|\mathrm{X} - \mathrm{E}(\mathrm{X})| \geq \varepsilon) \leq \mathrm{V}(\mathrm{X})/\varepsilon^2.$$

(Chernoff's inequality) Let $X_1, \ldots, X_n$ *be independent random variables taking values in* $\{0, 1\}$ *and let* $p_i = Pr(X_i = 1)$. *Then* $E(X) = p_1 + \cdots + p_n$ *for* $X = X_1 + \cdots + X_n$ *and for* $0 \leq \delta \leq 1$

$$\Pr(\mathrm{X} \leq (1 - \delta)\mathrm{E}(\mathrm{X})) \leq \mathrm{e}^{-\mathrm{E}(\mathrm{X})\delta^2/2}$$

and for all $\delta > 0$

$$\Pr(\mathrm{X} > (1 + \delta)\mathrm{E}(\mathrm{X})) \leq \left[\frac{\mathrm{e}^{\delta}}{(1 + \delta)^{1+\delta}}\right]^{\mathrm{E}(\mathrm{X})}$$

If $p_1 = \cdots = p_n = p$, X is binomially distributed and $E(X) = np$. The probability bounds are exponentially small with respect to n if δ is constant. The probability that a random string has less than $n/3$ ones can be bounded by $e^{-n/36}$ (choose $\delta = \frac{1}{3}$ implying that $(1-\delta)E(X) = n/3$).

The probability that one chosen bit flips in n^2 steps less than $n/2$ times can be bounded by $e^{-n/8}$ $(E(X) = n, \delta = 1/2)$. Even the probability that it flips less than $n - n^{3/4}$ times can be bounded by the asymptotically vanishing value of $e^{-n^{1/2}/2}$ $(E(X) = n, \delta = n^{-1/4})$. Hence, Chernoff's inequality gives concrete bounds (not only asymptotic ones) and it shows that the value of a binomially distributed random variable is with overwhelming probability very close to its expected value. This implies the following procedure. Produce an estimate by working in situations where Chernoff's inequality is applicable with expected values as "true" values. This leads to conjectures which often (with some weakening) can be proved rigorously.

6. The coupon collector's theorem

ONEMAX is perhaps the simplest function essentially depending on all variables and having a unique global optimum. Already Mühlenbein(1992) has proved an $O(n \log n)$ bound on the expected run time of the (1+1)EA on ONEMAX. This bound follows directly from a general upper bound technique presented in Section 8. What about a lower bound?

First, we present the coupon collector's theorem and then we argue why this result leads to an $\Omega(n \log n)$ lower bound on the expected run time of the (1+1)EA on ONEMAX. Consider the following experiment. There are n bins. Somebody throws randomly and independently balls into the bins and stops when each bin contains at least one ball. We are interested in the random number of balls in all bins together. The problem is equivalent to the problem of collecting coupons (pictures with, e. g., football players). There are n different types of coupons. We buy coupons until we have a complete collection of coupons. The expected time until we obtain a specific coupon equals n. However, we have to wait for the last coupon.

Theorem 3 *(Coupon collector's theorem)*
Let X be the random waiting time in the coupon collector's scenario. Then

$$\lim_{n\to\infty} Pr(X \leq n \ln n - cn) = e^{-e^c}$$

and

$$\lim_{n\to\infty} Pr(X \geq n \ln n + cn) = 1 - e^{-e^{-c}}.$$

The proof of this result is contained in Motwani and Raghavan (1995). The theorem is described as limit behavior. The bounds for sufficiently large n are close to the limit values. Such a result is also called a sharp

threshold result, since the constant factor for the leading term $n \ln n$ is given exactly and the probability even of small deviations is very small.

Now we apply this result to the analysis of the (1+1)EA on ONEMAX (Droste, Jansen, and Wegener (1998a)). The probability that the first search point has less than $2n/3$ ones is overwhelmingly large (Chernoff's bound). It is necessary that each 0-bit flips at least once. We investigate $n \ln n + cn$ steps and only $n/3$ bit positions which are initialized by 0. Again we can apply Chernoff's bound. With large probability, the number of flipping bits is bounded above by $\frac{1}{3} n \ln n + cn$. We consider the $n/3$ bit positions as bins and the flipping bits as balls. However, the balls are not totally independent. If during one step at least two bits flip, these bits are at different positions. It is possible to bound the difference between the real situation and the situation of the coupon collector's theorem. This also is a general rule. One often can apply results from probability theory — but not in their pure form.

Theorem 4 *The expected run time of the (1+1)EA on ONEMAX is bounded above by* $e \cdot n \cdot (\ln n + 1)$. *For each constant* $d > 0$ *there is a constant* c *such that the success probability of the (1+1)EA on a function with a unique global optimum within* $n \ln n - cn$ *steps is bounded above by* d.

With a more careful analysis one will even get better lower bounds for ONEMAX. However, the crucial fact is that the coupon collector's theorem leads to a lower bound for a large class of functions.

7. The gambler's ruin problem

The classical gambler's ruin problem is the following one. Alice owns a dollars and Bob owns b dollars. They are throwing coins and Alice pays Bob one dollar if the coin shows head and receives one dollar from Bob for tails. The game stops if somebody gets ruined. The problem is to compute the probability that Alice gets ruined and how long it does take until the game stops.

We may use ideas from the solution of the gambler's ruin problem in order to analyze the (1 + 1)EA on SPP, the short path with $n+1$ points where n of them have the same fitness (they belong to the plateau) and the last point is the global optimum. Since the function is defined as $n-$ONEMAX outside the path, it follows from the analysis of ONEMAX that the (1 + 1)EA needs an expected time of $O(n \log n)$ to reach the plateau. With overwhelming probability, the plateau is reached close to 0^n. Hence, we have to analyze the behavior of the (1 + 1)EA on the plateau. Only other points from the plateau and the global optimum are accepted. The situation is similar to the gambler's ruin problem —

with some differences. The probability of accepting a different search point equals $\Theta(\frac{1}{n})$ (otherwise, no bit is flipped or a search point outside the path is reached). The conditional probability of an accepted step of length 1 is $1-\Theta(\frac{1}{n})$, and, for $j \geq 2$, the conditional probability of a step length of j equals $\Theta((\frac{1}{n})^{j-1})$. One has to control with some technical effort the influence of steps of length $j \geq 2$. The steps of length 1 from 1^i0^{n-i}, $0 < i < n$, reach the points $1^{i-1}0^{n-i+1}$ and $1^{i+1}0^{n-i-1}$ each with probability 1/2 – like in the gambler's ruin problem. Nobody is ruined at 0^n, but all accepted new strings have more ones.

The main part of the analysis is easy. In order to generalize the result, we consider a plateau length l and a phase of length cnl^2, c a large enough constant. By Chernoff's bound, the number of accepted steps is with overwhelming probability at least $c'l^2$ (where c' can be chosen as any constant by increasing c). Here we ignore all accepted steps whose length is at least 2 (for these details see Jansen and Wegener, 2000b). We want to estimate the probability that the number of "increasing steps", i.e., steps of length 1 which increase the number of ones, is at least $c'l^2/2+l/2$. Such an event implies that we have reached the global optimum. By symmetry, the probability of at most $c'l^2/2$ increasing steps, equals 1/2. The binomial distribution with parameters m and 1/2 has its highest value at $\lfloor m/2 \rfloor$ where the probability is bounded above by $\alpha m^{-1/2}$ for some constant α. Hence, the probability of at least $c'l/2$ and at most $c'l^2/2+l/2$ increasing steps is bounded above by $(l/2+1)\cdot\alpha\cdot(c'l^2/2)^{-1/2}$ which can be bounded by 1/4 for large enough c'. This implies a success probability of at least 1/4 and the expected number of phases before a first success is at most 4. Hence, the expected time to pass a plateau which is a path of length l is bounded above by $O(nl^2)$. If the fitness is increasing along the path, the expected run time equals $\Theta(nl)$ (see Section 8). Obviously, it is easier to follow a path with increasing fitness, but even a path giving no hints is no real obstacle for the (1 + 1)EA. The following theorem describes the result for SPP where $l = n$.

Theorem 5 *The expected run time of the* (1 + 1)*EA on SPP is bounded above by* $O(n^3)$. *The success probability for* n^4 *steps is bounded below by* $1-2^{-\Omega(n)}$.

It is also possible to prove a matching lower bound on the expected run time.

8. Upper bounds by artificial fitness levels

The following upper bound technique is based on a suitable partition of the search space according to the fitness function.

Definition 13. For $A, B \subseteq \{0,1\}^n$ and $f\colon \{0,1\}^n \to \mathbb{R}$ the relation $A <_f B$ holds if $f(a) < f(b)$ for all $a \in A$ and $b \in B$. A $<_f$-partition of $\{0,1\}^n$ is a partition of $\{0,1\}^n$ into non-empty sets $A_1, \ldots, A_m$ such that $A_1 <_f A_2 <_f \cdots <_f A_m$ and all $a \in A_m$ are global optima.

Lemma 1 *Let $A_1, \ldots, A_m$ be a $<_f$-partition, let $p(A_i)$ be the probability that a randomly chosen search point belongs to A_i, let $s(a)$ be the probability that a mutation of $a \in A_i$ belongs to $A_{i+1} \cup \cdots \cup A_m$, and let $s_i = \min\{s(a) | a \in A_i\}$. Then*

$$E(X_f) \leq \sum_{1 \leq i \leq m-1} p(A_i)(s_i^{-1} + \cdots + s_{m-1}^{-1}) \leq s_1^{-1} + \cdots + s_{m-1}^{-1}.$$

The proof of Lemma 1 is very simple. The second inequality is trivial and the first inequality is based on the law of total probability. When starting in A_i, the expected time to leave A_i is bounded above by s_i^{-1} and we have to leave only some of the sets $A_i, \ldots, A_{m-1}$. Nevertheless, this result is surprisingly powerful. However, it is crucial to choose an appropriate $<_f$-partition. We start with simple applications leading to asymptotically optimal bounds.

For ONEMAX let A_i contain all points with i ones, $0 \leq i \leq n$. For $x \in A_i$ there are $n-i$ 1-bit mutations leading to A_{i+1} (if $i < n$). Hence, $s_i \geq (n-i)\frac{1}{n}(1-\frac{1}{n})^{n-1} \geq e^{-1} \cdot (n-i) \cdot \frac{1}{n}$ and

$$s_0^{-1} + \cdots + s_{n-1}^{-1} \leq e \cdot n \left(\frac{1}{1} + \frac{1}{2} + \cdots + \frac{1}{n}\right) \leq e \cdot n \cdot (\ln n + 1).$$

This bound can be improved by considering the probabilities $p(A_i)$, but the improvement saves only a linear additive term. A corresponding lower bound has been presented in Section 6.

For LO let A_i contain all points with fitness i, i.e., starting with i but not with $i+1$ ones. For $x \in A_i$ there is one 1-bit mutation leading to A_{i+1} (if $i < n$). Hence, $s_i \geq e^{-1} \cdot \frac{1}{n}$ and we obtain the upper bound $e \cdot n^2$ where l is the length of the path. A corresponding lower bound will be presented in Section 9.

For path functions we assume that the values outside the path are chosen in such a way that the path is reached quickly (e.g., n–ONEMAX if 0^n is the source of the path). The expected run time for the first point on the path is denoted by $t(n)$. For the rest of the search let A_i contain the ith point of the path. Since there is by definition a 1-bit mutation to the successor on the path, we get an upper bound for the expected run time of size $e \cdot n \cdot l + t(n)$. A corresponding lower bound will be presented in Section 9.

These considerations can be generalized to all unimodal functions, since there is always a 1-bit mutation improving the fitness. If the unimodal function f takes $w(f)$ different values, the expected run time of the algorithm can be bounded above by $e \cdot n \cdot (w(f) - 1)$. The function BV (binary value) is unimodal and takes the maximal number of 2^n different values. The upper bound $e \cdot n \cdot (2^n - 1)$ on the expected run time is correct. However, this bound seems to be far from optimal. Sitting in 01^{n-1}, it is very likely that the next accepted step improves the fitness by much more than 1.

We investigate the class of degree-k functions f with N non-vanishing weights which are all positive (Wegener and Witt, 2000). We number the N positive weights in non-increasing order: $w_1 \geq w_2 \geq \cdots \geq w_N > 0$. Then we use the following $<_f$-partition $A_0, \ldots, A_N$ where

$$A_i = \{x | w_1 + \cdots + w_i \leq f(x) < w_1 + \cdots + w_{i+1}\}$$

for $0 \leq i \leq N - 1$ and

$$A_N = \{x | f(x) = w_1 + \cdots + w_N\}.$$

For $x \in A_i$ there is some $j \in \{1, \ldots, i+1\}$ such that the w_j-term is not activated, i.e., not all bits belonging to S_j are equal to 1. The mutation where all 0-bits belonging to S_j flip activates the w_j-term. This increases the f-value by at least w_j. Here it is essential that all weights are non-negative. By definition of A_i, the resulting search point belongs to $A_{i+1} \cup \cdots \cup A_N$. Since we consider a mutation of at most $|S_j| \leq k$ bits, $s_i^{-1} \leq e \cdot n^k$ leading to an upper bound on the expected run time of $e \cdot n^k \cdot N$. For linear functions, negative weights can be replaced with positive weights without changing the behavior of the (1+1)EA. Variables x_i with $w_i < 0$ are replaced with $\overline{x}_i = 1 - x_i$. For degree-k functions, $k \geq 2$, such a replacement can create new negative weights. Since $N \leq n$ for linear functions, we obtain an $e \cdot n^2$ upper bound for all linear functions and the upper bound for BV is decreased from exponential to quadratic. An even better upper bound will be presented in Section 10. We summarize our results.

Theorem 6 *The following upper bounds hold for the expected run times of the (1+1)EA:*

- $e \cdot n \cdot (\ln n + 1)$ *for ONEMAX,*
- $e \cdot n^2$ *for all linear functions,*
- $e \cdot n^k \cdot N$ *for all degree-k functions with N non-vanishing weights which all are positive,*

- $e \cdot n^2$ *for LO,*
- $e \cdot n \cdot l$ *for path functions on paths of length* l *if the search starts on the path,*
- $e \cdot n \cdot N$ *for all unimodal functions taking at most* $N + 1$ *different values.*

9. Lower bounds by artificial fitness levels

We look for a lower bound result corresponding to the upper bound of Lemma 1. We use the notations of Lemma 1. If the initial search point belongs to A_i, $i < m$, the search has to leave A_i. Let $\mu_i = \max\{s(a) | a \in A_i\}$. Then the expected time to leave A_i is bounded below by μ_i^{-1} leading to the following result.

Lemma 2 $E(X_f) \geq \sum_{1 \leq i \leq m-1} p(A_i)\mu_i^{-1}$.

Lemma 2 is less powerful than Lemma 1. The reason is that we usually do not reach a global optimum from A_i. Hence, we should investigate how many A-levels we usually pass in one step. Such an approach has been realized for LO and path functions by Droste, Jansen, and Wegener (1998b).

For LO we investigate the stochastic process behind the (1 + 1)EA more carefully. If we have produced a string x where $\mathrm{LO}(x) = i$, then $x_1 = \cdots = x_i = 1$, $x_{i+1} = 0$, and $(x_{i+2}, \ldots, x_n)$ is a random string. The last property is easy to prove and essential. If a step increases the fitness, we know that none of the first i bits is flipping, the $(i + 1)$st bit flips and the new suffix $(x'_{i+2}, \ldots, x'_n)$ again is a random string. We have k "free-riders" if exactly k leading bits of $(x'_{i+2}, \ldots, x'_n)$ are ones. The production of free-riders can be described by the following stochastic process. We have a random bit string $a = (a_1, a_2, \ldots)$. The number of ones between two consecutive zeros describes the number of free-riders. The probability that we have more than $2n/3$ free-riders during $n/3$ fitness-increasing steps (including the initialization) is equal to the probability that a random string from $\{0,1\}^n$ contains more than $2n/3$ ones. By Chernoff's bound this probability is exponentially small. Hence, we have to wait with overwhelming probability for at least $n/3$ fitness-increasing steps. The probability of a fitness-increasing step is bounded above by $1/n$, since one special bit has to flip. Another application of Chernoff's bound shows that with overwhelming probability $n^2/6$ steps are not enough to have $n/3$ fitness-increasing steps.

In the following we investigate the long path function LP. It is sufficient to know that the path length is $l = \Theta(n^{1/2}2^{n^{1/2}})$. Each point p_i on

the path has for each $d \leq n^{1/2}$ at most one successor p' on the path such that $H(p_i, p') = d$. If $i + d \leq l$, $H(p_i, p_{i+d}) = d$. Moreover, the fitness outside the path is defined in such a way that we can assume that the path is reached for the first time in its first half. The idea is to estimate the expected progress along the path during one step.

The probability that at least $n^{1/2}$ bits flip simultaneously, can be bounded by $2^{-\Omega(n^{1/2}\log n)}$. For such cases we estimate the progress by the simple upper bound l leading to a contribution of $2^{-\Omega(n^{1/2}\log n)}$ to the expected progress. If less than $n^{1/2}$ bits flip simultaneously, only one special k-bit mutation is accepted and leads to a progress of k on the path. The probability of a special k-bit mutation is bounded above by n^{-k}. Hence, the expected progress in one step is bounded above by

$$\sum_{1 \leq k < n^{1/2}} k \cdot n^{-k} + 2^{-\Omega(n^{1/2}\log n)} \leq \frac{2}{n} + 2^{-\Omega(n^{1/2}\log n)}.$$

The expected progress within $l \cdot n/5$ steps is bounded above by $(\frac{2}{5} + o(1))l$ and we need a progress of $l/2$. Markov's inequality proves a bound of $\Omega(l \cdot n)$ on the expected run time of the (1 + 1)EA on LP.

Theorem 7 *The following lower bounds hold on the expected run times of the* (1 + 1)*EA:*

- $n^2/6 - o(n^2)$ *for LO.*
- $\Omega(ln)$ *for long path functions where* $l = O(n^{1/2}2^{n^{1/2}})$ *not allowing short cuts by at most* $n^{1/2}$ *flipping bits and the property that the path is reached with constant positive probability in the first half.*

Such a long path function where $l = O(n^{1/2}2^{n^{1/2}})$ has been defined proving that unimodal functions can be hard for the (1 + 1)EA.

10. Potential functions

The general upper bound for unimodal functions leads to a bad upper bound for BV (binary value). The design of a problem-specific $<_f$-partition allows a simple proof of a quadratic upper bound. The same holds for all linear functions. We have a gap between the $\Omega(n \log n)$ lower bound from Section 6 and the $O(n^2)$ upper bound from Section 8. Droste, Jansen and Wegener (1998a) have improved the upper bound to $O(n \log n)$. It is difficult to control the Hamming distance to the global optimum which, under the assumption $w_1 \geq \cdots \geq w_n > 0$, equals 1^n. Hence, the idea is to measure the progress with respect to some well-chosen measure. The idea of $<_f$-partitions is already a step into

this direction. These partitions have the advantage that only strings from $A_i \cup \cdots \cup A_m$ can be accepted from $x \in A_i$. Hence, the function $g(t)$ describing the index of the A-set containing the actual search point after t steps is a non-decreasing function. The asymptotically exact bound for linear functions has been obtained only by using a so-called potential function where it is quite likely that the value of the potential decreases in some steps. In particular, these potential functions do not depend on the special values of the linear function f.

The new method is easier to explain for BV. The potential function equals ONEMAX, the number of ones in the string. To be precise, the following is important:

- the (1+1)EA uses the fitness function f in order to decide whether the old search point is replaced with its mutant,
- the people analyzing the (1+1)EA use the potential function p to measure the progress.

A step is called successful if the mutant x' is different from its parent x and replaces x. The crucial step is to estimate the expected number of steps until the (1+1)EA produces from x with $p(x) = i$ a search point x' with a higher p-value. We distinguish between successful and unsuccessful steps. Knowing the expected number of successful steps it is in this special situation not too difficult to bound the expected number of unsuccessful steps.

BV has the advantage that a successful step has a simple description. A step is successful iff the leftmost flipping bit is a 0. All other bits flip with mutation probability $1/n$ even under the assumption that the step is successful. Knowing that x contains i ones we do not know how many are to the right of the leftmost flipping bit which is a zero. Hence, we pessimistically analyze a provable slower Markov process assuming that only the leftmost flipping bit is a flipping 0 and that $n - i$ 1-bit positions have the chance to flip to 0. Now we run into difficulties. The expected progress with respect to the potential function can only be bounded below by $1 - \frac{n-i}{n} = \frac{i}{n}$ leading to an expected number of at most n/i successful steps before the p-value has increased. The last conclusion follows from Wald's identity. For BV, a simple trick works. We restrict our attention to the first $n/2$ partitions. The behavior of the second half of the n bits has no influence on the decision whether the actual first half of the string is changed by the (1+1)EA. The expected progress is now bounded below by 1/2 leading to an expected number of at most two successful steps before the p-value has increased. This leads after some calculations on the number of unsuccessful steps to an $O(n \log n)$ bound for the time until the actual search point starts with

$n/2$ ones. The second half can be treated in the same way. Only the number of unsuccessful steps has to be multiplied by $(1-\frac{1}{n})^{n/2} \approx e^{-1/2}$, the probability that no bit of the first half flips. This describes the proof method for the special case of BV. There are many places where the very special properties of BV have been used.

Surprisingly, there is one special linear function serving as a potential function for all linear functions. This function is

$$p(x) = 2 \cdot (x_1 + \cdots + x_{n/2}) + 1 \cdot (x_{n/2+1} + \cdots + x_n).$$

This potential function is somehow a compromise between equal weights (ONEMAX) and very different weights (BV). By an involved and tedious case inspection, it can be shown that the expected number of successful steps until the value of the potential function p has increased is bounded above by a constant (independent from the starting point). Then the methods discussed for the analysis of the (1+1)EA on BV can be generalized to lead to the proposed bound.

Theorem 8 *The expected run time of the (1+1)EA on an arbitrary linear function is bounded above by* $O(n \log n)$.

11. Investigations of typical runs

The sections on tail inequalities and the coupon collector's theorem have shown that stochastic experiments or stochastic processes have a "typical global behavior", although the local behavior is unpredictable. Tossing coins one has no idea about the outcome of a toss. Tossing independently many coins we also have no idea about a certain coin, but we have very tight bounds describing the number of heads — if we allow an exponentially small error probability. The same is true for the coupon collector. Obtaining the next coupon she or he can only hope for a coupon which she or he does not hold yet. Obtaining $g(n)$ coupons, for each value of $g(n)$ outside a small interval one can be almost sure to obtain all different coupon types or one can be almost sure that some coupons are missing. The same holds for evolutionary algorithms. The behavior within one step or a few steps has a large uncertainty, although one can obtain bounds for the global behavior which have a very small error probability.

The idea is to investigate search phases of well-chosen lengths. During each phase we expect a certain behavior of the (1+1)EA. Runs of the (1+1)EA fulfilling the aims of all phases are called "typical". We should choose our definition in a way that the run time of a typical run has properties we expect to happen with overwhelming probability. The essential point is to prove an upper bound on the failure probability, i.e.,

the probability of a non-typical run. This can be done by adding the failure probabilities for the different phases. Sometimes, we get better results if we ensure that the different failure events are independent. Otherwise, we can bound the failure probability for the ith phase using the assumption that the first $i-1$ phases were free of failures.

A simple application of this method is the analysis of the (1+1)EA on TRAP. The simple upper bound n^n holds for all pseudo-boolean functions (Droste, Jansen, and Wegener, 1998b). We expect that this bound is quite tight for TRAP, since TRAP gives everywhere (except at the global optimum) wrong hints. However, the initial search point has approximately $n/2$ ones and the probability that mutation creates the global optimum is approximately $n^{-n/2}$. The expected waiting time for an event with such a success probability is approximately $n^{n/2}$ – only the square root of n^n.

Now we consider a typical run consisting of three phases:

- the initialization phase, we expect an initial string with less than (2/3) ones,
- the phase of the first $cn^2 \log n$ steps, we expect to have 0^n as last actual search point,
- the phase of the remaining steps starting with 0^n, the expected run time equals n^n.

Without failures during the first two phases we have 0^n as actual point and accept only 1^n as new search point. The probability that mutation produces 1^n from 0^n equals n^{-n} leading to an expected waiting time of n^n. The failure probability of the first phase is $2^{-\Omega(n)}$ (Chernoff's bound). If there was no failure in the first phase, a failure in the second phase implies that we either flip at least $n/3$ steps in one step (failure type 1) or the optimization of $n-$ ONEMAX takes more than $cn^2 \log n$ steps (failure type 2). The probability of a failure of type 1 in one step is bounded above by $\frac{1}{(n/3)!} = 2^{-\Omega(n \log n)}$ leading to an upper bound of $cn^2(\log n)2^{-\Omega(n \log n)} = 2^{-\Omega(n \log n)}$ for the whole phase. We know that the expected run time of the (1+1)EA on $n-$ ONEMAX is bounded above by $c'n \log n$ for some constant c'. Let $c = 2c'$. Then we get n independent subphases of length $2c'n \log n$ each. The probability that a subphase is unsuccessful is bounded above by 1/2 (Markov's inequality) leading to a 2^{-n} bound for the probability of type 2 failures. (It is possible to obtain even better bounds.) Altogether, with probability $1-2^{-\Omega(n)}$ no failure occurs leading to an expected run time of n^n. This proves the following result.

Theorem 9 *The expected run time of the (1+1)EA on TRAP is bounded above by* n^n *and below by* $(1 - 2^{-\Omega(n)})n^n$.

A more sophisticated application of this method has been presented by Jansen and Wegener (1999). Crossover is known as one of the fundamental operators of evolutionary algorithms. Nevertheless, Jansen and Wegener (1999) were the first to prove rigorously for a function, namely JUMP^m for $m = \lfloor \log n \rfloor$, that the expected run time of evolutionary algorithms without crossover grows superpolynomially $(n^{\Omega(\log n)})$ while a steady-state genetic algorithm with population size n and the small crossover probability $p_c = 1/(n \log n)$ has a polynomial expected run time.

Theorem 10 *Each evolutionary algorithm without crossover needs for* $JUMP^m$, $m = \lfloor \log n \rfloor$, *with large probability superpolynomial time. A steady-state genetic algorithm with population size* n *and probability* $p_c = 1/(n \log n)$ *for uniform crossover has an expected run time on* $JUMP^m$ *which is bounded above by* $O(n^3 \log n)$.

We discuss the problems with the proof of an upper bound for the steady-state EA. It seems easy to obtain a population consisting only of individuals with exactly m zeros (or even an optimal individual). Uniform crossover on two individuals which have m zeros each and the zeros at different positions has a probability of exactly $2^{-2m} \geq 1/n^2$ to produce the global optimum 1^n and the probability that mutation does not destroy 1^n is approximately e^{-1}. The expected waiting time increases to $O(n^3 \log n)$ because of the small crossover probability. This small probability simplifies the control of the hitchhiking effect. If we choose two random strings with m zeros, it is very likely to have the zeros at different positions. Crossover and selection destroy the independency of the individuals of the initial population. It gets more likely that individuals share zeros. The proof of Theorem 10 is possible with the following definition of a typical run. Since the estimation of the failure probabilities is too complicated to be presented here in detail, we also do not present the chosen phase lengths.

We expect that after the first phase either we have found the optimum or that all individuals have exactly m zeros. The failure probability can be estimated using a generalization of the coupon collector's theorem and Chernoff's bound. After the second phase we expect that either we have found the optimum or all individuals still have exactly m zeros and for each bit position the number of individuals with a zero at this position is bounded above by $n/(4m)$. We sum the exponentially small failure probabilities for the n different bit positions. The small crossover

probability makes it possible to consider crossover as a bad event which increases the number of individuals with a zero at the considered position. For the third phase we expect that for no bit position and no point of time the number of individuals with a zero gets larger than $n/(2m)$. This implies a probability of at least 1/2 that two individuals have no zero in common and crossover can work. Hence, we expect to find the optimum during the third phase.

Conclusion

The analysis of evolutionary algorithms can be based on methods from the analysis of classical deterministic and randomized algorithms. Some tools which seem to be of large value for evolutionary algorithms have been presented, discussed, and applied. This has led to a number of results on the expected run time and the success probability of evolutionary algorithms.

References

Bäck, T. (1993). Optimal mutation rates in genetic search. Proc. of 5th ICGA (Int. Conf. on Genetic Algorithms), 2–8.

Bäck, T. (1998). An overview of parameter control methods by self-adaptation in evolutionary algorithms. Fundamenta Informaticae 35, 51–66.

Bäck, T., Fogel, D. B., and Michalewicz, Z. (Eds). (1997). *Handbook of Evolutionary Computation.* Oxford Univ. Press.

Beyer, H.-G. (1996). Toward a theory of evolution strategies: Selfadaptation. Evolutionary Computation 3, 311–347.

Cormen, T. H., Leiserson, C. E., and Rivest, R. L. (1990). *Introduction to Algorithms.* MIT Press.

Droste, S., Jansen, T., and Wegener, I. (1998a). A rigorous complexity analysis of the (1+1) evolutionary algorithm for separable functions with Boolean inputs. Evolutionary Computation 6, 185–196.

Droste, S., Jansen, T., and Wegener, I. (1998b). On the optimization of unimodal functions with the (1+1) evolutionary algorithm. Proc. of PPSN V (Parallel Problem Solving from Nature), LNCS 1498, 13–22.

Fogel, D. B. (1995). *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence.* IEEE Press.

Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison Wesley.

Hochbaum, D. S. (1997). *Approximation Algorithms for NP-Hard Problems.* PWS Publ. Co., Boston.

Holland, J. H. (1975). *Adaptation in Natural and Artificial Systems.* The University of Michigan Press.

Horn, J., Goldberg, D. E., and Deb, K. (1994). Long path problems. Proc. of PPSN III (Parallel Problem Solving from Nature), LNCS 866, 149–158.

Jansen, T., and Wegener, I. (1999). On the analysis of evolutionary algorithms – a proof that crossover really can help. Proc. of ESA '99 (European Symp. on Algorithms), LNCS 1643, 184–193.

Jansen, T., and Wegener, I. (2000a). On the choice of the mutation probability for the (1+1)EA. Proc. of PPSN VI (Parallel Problem Solving from Nature), LNCS 1917, 89–98.

Jansen, T., and Wegener, I. (2000b). Evolutionary algorithms — how to cope with plateaus of constant fitness and when to reject strings of the same fitness. Submitted to IEEE Trans. on Evolutionary Computation.

Motwani, R., and Raghavan, P. (1995). *Randomized Algorithms.* Cambridge Univ. Press.

Mühlenbein, H. (1992). How genetic algorithms really work. I. Mutation and hillclimbing. Proc. of PPSN II (Parallel Problem Solving from Nature), 15–25.

Rabani, Y., Rabinovich, Y., and Sinclair, A. (1998). A computational view of population genetics. Random Structures and Algorithms 12, 314–334.

Rudolph, G. (1997). How mutations and selection solve long path problems in polynomial expected time. Evolutionary Computation 4, 195–205.

Schwefel, H.-P. (1995). *Evolution and Optimum Seeking.* Wiley.

Wegener, I., and Witt, C. (2000). On the behavior of the (1+1) evolutionary algorithm on quadratic pseudo-boolean functions. Submitted to Evolutionary Computation.

This page intentionally left blank

Chapter 15

A GENETIC ALGORITHM HEURISTIC FOR FINITE HORIZON PARTIALLY OBSERVED MARKOV DECISION PROBLEMS

Alex Z.-Z. Lin,
James C. Bean and
Chelsea C. White III

Abstract The partially observed Markov decision problem (**POMDP**) is a generalization of a Markov decision problem (**MDP**) that allows for noise corrupted and costly observations of the underlying system state. The value function of the **POMDP** is known to be piecewise affine and convex (**PAC**) in the probability mass vector (**pmv**) over the state space. Most exact solution procedures determine the minimal set of affine functions that describes the value function. This determination tends to require significant computational resources, and as a result these solution procedures can only solve small-scale problems.

In this paper, we develop a heuristic approach, DC-NICHE, for constructing suboptimal designs for the finite horizon **POMDP** and present error bounds. We use the genetic algorithm to construct approximations of the minimal set of affine functions that describes the value function. Numerical results indicate that 1) these heuristics can quickly yield high quality solutions for a set of small-scale problems where exact solutions are feasible; and 2) can provide suboptimal designs for realistically sized problems.

1. Introduction

This paper presents a procedure for constructing a suboptimal strategy for the finite horizon partially observed Markov decision problem (**POMDP**) with finite state, action, and observation sets. This procedure uses a genetic algorithm (**GA**) and is based on the fact that

the optimality equation for the **POMDP** is piecewise affine and convex (**PAC**) in the probability mass vector (**pmv**) over the state space. This procedure permits the consideration of relatively large, realistically sized problems.

The (completely observed) Markov decision problem (**MDP**) is a model of sequential decision making under uncertainty which assumes that the decision-maker has access to the exact value of the current state of the system without cost. Surveys of the **MDP** can be found in (White, 1985; White, 1988; White, III and White, 1989). The **POMDP** is an extension of the **MDP** that relaxes this assumption and hence permits costly and/or noise corrupted state observations. For example, consider the situation where a physician selects diagnostic and/or treatment decisions on the basis of the patient's signs, symptoms and lab test results, rather than on the patient's underlying, and likely unobservable, state of health. Other examples can be found in (Smallwood and Sondik, 1973; White, III and White, 1989).

Relative to the **MDP**, the enhanced validity of the **POMDP** is at the expense of its tractability in two ways. First, the **POMDP** is more data intensive than the **MDP**, additionally requiring information regarding how observations are related to state and action values. Second, current numerical solution procedures for the **POMDP** are not tractable for realistically sized problems. See (Lark, 1989; Lovejoy, 1991; Monahan, 1982; Platzman, 1980; Smallwood and Sondik, 1973; Sondik, 1971; Sondik, 1978; White, III and Scherer, 1989; White, III and Scherer, 1994) for further discussions. In this paper we partially address this limitation.

This chapter is organized as follows. In Section 2, the **POMDP** is defined and several fundamental results are presented on which our heuristics are based. The basics of **GA** are discussed in Section 3 within the context of the **POMDP.** Section 4 reviews the concept of random keys and presents the **GA**-based heuristic *DC-NICHE.* Procedures for reducing the complexity of the problem are presented. Error bounds are discussed in Section 5. The numerical evaluation in Section 6 compares the **GA**-based heuristic with two other algorithms. These results indicate that DC-NICHE performs especially well. Section 7 summarizes research results.

2. Partially Observed MDP

2.1 Definition

Let **S**, **A**, and **Z** be finite sets representing the *state, action,* and *observation* spaces, respectively. Let $\{0, 1, \ldots, T-1\}$ be the set of *stages* (or

epochs), where the *problem horizon* T is finite ($T < \infty$). Assume the *state* process $\{s(t),\ t = 0, 1, \ldots, T\}$, *action* process $\{a(t),\ t = 0, 1, \ldots, T-1\}$, and *observation* process $\{z(t),\ t = 1, 2, \ldots, T-1\}$ are linked by the probabilities $\mathbf{P}(z, a) = \{p_{ij}(z, a)\}$, where

$$p_{ij}(z, a) = \mathrm{P}[s(t+1) = j, z(t+1) = z \mid s(t) = i,\ a(t) = a].$$

Thus, if the underlying state of the process is $s(t)$ and the action selected is $a(t)$ at stage t, then the underlying state makes transition to state $s(t+1)$ and $z(t+1)$ is observed at stage $t+1$, according to the probability kernel $\mathbf{P}(z, a)$.

Let $r(i, a)$ be the reward accrued at stage t, given $s(t) = i$ and $a(t) = a$, and let $\bar{r}(i)$ be the reward received at terminal stage T, given $s(T) = i$. We assume the action at stage t is selected on the basis of the *history* at stage t, which is comprised of an a priori **pmv** on **S**, all past and present observations, and all past actions selected. A function mapping the set of histories at stage t into the action set is a *policy* at stage t; a sequence of policies, one for each stage $t = 0, 1, \ldots, T-1$, is a *strategy*. The problem *objective* is to determine a strategy that maximizes the total expected discounted reward to be accrued over the problem horizon, relative to the set of all strategies, conditioned on the a priori **pmv.**

2.2 Preliminary Results

Let $\mathbf{X} = \{\mathbf{x} :\ x_i \geq 0,\ i \in \mathbf{S},\ \sum_{i \in \mathbf{S}} x_i = 1\}$. Define $x(t) = \{x_i(t),\ i \in \mathbf{S}\}$, where $x_i(t)$ is the probability that $s(t) = i$, given the history at stage t. It is shown in (Smallwood and Sondik, 1973; White, III, 1991) that the resulting process, the *information* process $\{\mathbf{x}(t),\ t = 0, 1, \ldots, T\}$, is a *sufficient statistic* for the **POMDP.** Hence, it is sufficient to base action selection at stage t on $\mathbf{x}(t)$. Let $v_t(\mathbf{x})$ be the maximum expected total discounted reward to be accrued from stage t through the terminal stage T, given $\mathbf{x}(t) = \mathbf{x}$. Then it has been shown in (White, III, 1991) and elsewhere that the following optimality equation and boundary condition hold:

$$\begin{aligned} v_t(\mathbf{x}) &= \max_{a \in \mathbf{A}} \left\{\mathbf{x}\mathbf{r}(a) + \beta \textstyle\sum_{z \in \mathbf{Z}} \sigma(z, \mathbf{x}, a) v_{t+1}[\lambda(z, \mathbf{x}, a)]\right\}, \\ v_T(\mathbf{x}) &= \mathbf{x}\bar{\mathbf{r}}, \end{aligned} \tag{15.1}$$

where $\sigma(z, \mathbf{x}, a) = \mathbf{x}\mathbf{P}(z, a)\underline{1}$, $\underline{1}$ *is a vector of ones,* $\lambda(z, \mathbf{x}, a) = \mathbf{x}\mathbf{P}(z, a)/\sigma(z, \mathbf{x}, a)$, $\sigma(z, \mathbf{x}, a) \neq 0$, $\bar{\mathbf{r}} = \{\bar{r}(i),\ i \in \mathbf{S}\}$ and $\beta \geq 0$ is the discount factor. We remark that $\lambda(z, \mathbf{x}, a)$ is the **pmv** at stage $t+1$ when $z(t+1) = z$, $\mathbf{x}(t) = \mathbf{x}$, and $a(t) = a$, and $\sigma(z, \mathbf{x}, a)$ is the probability of observing z at stage $t+1$, given that $\mathbf{x}(t) = \mathbf{x}$ and $a(t) = a$. An optimal strategy is composed of policies which for stage t selects an action that

causes the maximum to be attained in (15.1) when $\mathbf{x}(t) = \mathbf{x}$ (White, III, 1991). Hence, solving the optimality equation for $t = 0, 1, \ldots, T$ is key to determining an optimal strategy. We now examine structural results that lead to the solution of the optimality equation.

2.3 Structural Results

We say that a real-valued function f on $\mathbf{X}$ is piecewise affine and convex (**PAC**) if and only if there exists a finite set Γ such that $f(\mathbf{x}) = \max\{\mathbf{x}\gamma : \gamma \in \Gamma\}$. The foundation of our approach is the fact that if v_{t+1} is **PAC**, then v_t is **PAC**, and since v_T is **PAC**, v_t is **PAC** for all t (White, III, 1991). We endeavor to determine v_t from v_{t+1} by determining Γ_t, given Γ_{t+1}, where $v_t(\mathbf{x}) = \max\{\mathbf{x}\gamma : \gamma \in \Gamma_t\}$ and $v_{t+1}(\mathbf{x}) = \max\{\mathbf{x}\gamma : \gamma \in \Gamma_{t+1}\}$. We now show how such a set Γ_t can be constructed, given Γ_{t+1}. Define $\mathbf{G}(\Gamma) = \bigcup_{a\in\mathbf{A}} \{\mathbf{r}(a) + \beta \sum_{z\in\mathbf{Z}} \mathbf{P}(z,a)\gamma^z : \gamma^z \in \Gamma\}$. Note that

$$\begin{aligned} v_t(\mathbf{x}) &= \max_{a\in\mathbf{A}}\{\mathbf{x}\mathbf{r}(a) + \\ &\qquad \beta \textstyle\sum_{z\in\mathbf{Z}} \sigma(z,\mathbf{x},a) \max[\lambda(z,\mathbf{x},a)\gamma : \gamma \in \Gamma_{t+1}]\} \\ &= \max_{a\in\mathbf{A}} \{\mathbf{x}\mathbf{r}(a) + \beta \textstyle\sum_{z\in\mathbf{Z}} \max[\mathbf{x}\mathbf{P}(z,a)\gamma : \gamma \in \Gamma_{t+1}]\} \\ &= \max\{\mathbf{x}\gamma : \gamma \in \mathbf{G}(\Gamma_{t+1})\}. \end{aligned}$$

That is, $\mathbf{G}$ is a recursion operator on a set of affine functions defining a value function.

2.4 PURGE Operator

Thus, we could set $\Gamma_t = \mathbf{G}(\Gamma_{t+1})$. However, $\mathbf{G}(\Gamma_{t+1})$ usually contains many *redundant* vectors, where γ is redundant if $v_t(\mathbf{x})$ is strictly greater than $\mathbf{x}\gamma$ for all $\mathrm{x} \in \mathbf{X}$. From both storage and computational perspectives, there is value in keeping the cardinality of Γ_t as small as possible and hence removing as many redundant vectors from $\mathbf{G}(\Gamma_{t+1})$ as possible in order to define Γ_t. Let the operator **PURGE** be such that $\Gamma' = \mathbf{PURGE}(\Gamma)$ is the subset of Γ having the smallest cardinality that satisfies $\max\{\mathbf{x}\gamma : \gamma \in \Gamma'\} = \max\{\mathbf{x}\gamma : \gamma \in \Gamma\}$ for all $\mathrm{x} \in \mathbf{X}$. Each element in set Γ' is referred to as a *defining* vector. Existence of such a subset is assured by results in (Littman et al., 1995). Then ideally we would want to select $\Gamma_t = \mathbf{PURGE}(\mathbf{G}(\Gamma_{t+1}))$.

Further examination of the **PURGE** operation will prove useful. Define

1 *For two sets* Γ_1 *and* Γ_2, *let* $\Gamma_1 \oplus \Gamma_2 = \{\gamma_1 + \gamma_2 : \gamma_1 \in \Gamma_1,\ \gamma_2 \in \Gamma_2\}$.

2 *For scalar* β *and set* Γ, *let* $\beta\Gamma = \{\beta\gamma : \gamma \in \Gamma\}$.

3 *For vector* γ' *and set* Γ, *let* $\gamma' + \Gamma = \{\gamma' + \gamma : \gamma \in \Gamma\}$.

It is straightforward to show that for $\circ \in \{+, \cup\}$ and for $K \geq 2$,

$$\mathbf{PURGE}(\mathbf{Y}_1 \circ \cdots \circ \mathbf{Y}_K) = \mathbf{PURGE}(\mathbf{PURGE}(\mathbf{Y}_1 \circ \cdots \circ \mathbf{Y}_{K-1}) \circ \mathbf{Y}_K), \tag{15.2}$$

where $\mathbf{Y}_k, k = 1, \ldots, K$, are given sets. These properties suggest the following decompositions:

$$\Gamma_{t+1}^{z,a} = \mathbf{PURGE}\left(\{\mathbf{P}(z,a)\gamma : \gamma \in \Gamma_{t+1}\}\right), \tag{15.3}$$

$$\Gamma_t^a = \mathbf{PURGE}\left(\left\{\mathbf{r}(a) + \beta \sum_{z \in \mathbf{Z}} \gamma^z : \gamma^z \in \Gamma_{t+1}^{z,a}\right\}\right), \tag{15.4}$$

$$\Gamma_t = \mathbf{PURGE}\left(\bigcup_{a \in \mathbf{A}} \Gamma_t^a\right). \tag{15.5}$$

We remark that the cardinality of $\Gamma_{t+1}^{z,a}$ is usually much smaller than the cardinality of Γ_{t+1}, and hence the **PURGE** step described in (15.3) is especially useful in removing redundant vectors. As noted in (Eagle, 1984), a procedure we also have found that is useful for quickly removing redundant vectors in $\{\mathbf{P}(z,a)\gamma : \gamma \in \Gamma\}$ is: remove γ'' from Γ if there is a $\gamma' \in \Gamma$ such that $\gamma_i' \geq \gamma_i''$ for all $i \in \mathbf{S}$.

2.5 Ψ Operator

We now introduce the operator Ψ that will be useful in describing our heuristic. For any $\mathbf{X}' \subseteq \mathbf{X}$, let $\Psi(\Gamma', \mathbf{X}') = \{\Psi(\Gamma', \mathbf{x}') : \mathbf{x}' \in \mathbf{X}'\}$, where $\Psi(\Gamma', \mathbf{x}') = \gamma'$ if and only if (i) $\gamma' \in \mathbf{G}(\Gamma')$ and (ii) $\mathbf{x}'\gamma' \geq \mathbf{x}'\gamma \ \forall \ \gamma \in \mathbf{G}(\Gamma')$. Note that $\Psi(\Gamma, \mathbf{X}) = \mathbf{PURGE}(\mathbf{G}(\Gamma))$. Ideally, we seek the collection of sets $\{\Gamma_t, t = 0, 1, \ldots, T\}$, where $\Gamma_t = \Psi(\Gamma_{t+1}, \mathbf{X})$ and $\Gamma_{T,T} = \{\bar{\mathbf{r}}\}$. The difficulty with determining $\Psi(\Gamma, \mathbf{X})$ is that $\mathbf{X}$ is uncountable. The following result provides further insight into the properties of G and Ψ. The proof is straightforward. Let $\Gamma'' \prec \Gamma'$ if and only if $\max\{\mathbf{x}\gamma'' : \gamma'' \in \Gamma''\} \leq \max\{\mathbf{x}\gamma' : \gamma' \in \Gamma'\}$ for all $\mathbf{x} \in \mathbf{X}$.

Lemma 1

1 If $\Gamma'' \subseteq \Gamma'$, *then* $\Gamma'' \prec \Gamma'$.

2 If $\Gamma'' \prec \Gamma'$, *then* $\Psi(\Gamma'', \mathbf{X}') \prec \Psi(\Gamma', \mathbf{X}'), \ \forall \ \mathbf{X}' \subseteq \mathbf{X}$.

3 If $\Gamma'' \subseteq \Gamma'$, *then* $\mathbf{G}(\Gamma'') \subseteq \mathbf{G}(\Gamma')$.

4 If $\mathbf{X}'' \subseteq \mathbf{X}' \subset \mathbf{X}$, *then* $\Psi(\Gamma, \mathbf{X}'') \subseteq \Psi(\Gamma, \mathbf{X}')$.

2.6 Determination of $\Psi(\Gamma,\bullet)$

The key result to determine $\Psi(\Gamma, \mathbf{x})$ on which our heuristic is based is

Lemma 2 *Given* Γ_{t+1} *and arbitrary* $\mathbf{x}' \in \mathbf{X}$, *there exists a* $\gamma' \in \Gamma_t$ *such that* $v_t(\mathbf{x}') = \mathbf{x}'\gamma'$ *and an optimal action for stage* t, *assuming* $\mathbf{x}(t) = \mathbf{x}'$.

Proof: For each $z \in \mathbf{Z}$ and $a \in \mathbf{A}$, let $\gamma(z,a) \in \Gamma_{t+1}^{z,a}$ be such that $\mathbf{x}'\gamma(z,a) \geq \mathbf{x}'\gamma$ for all $\gamma \in \Gamma_{t+1}^{z,a}$. Let $a' \in \mathbf{A}$ be such that for all $a \in \mathbf{A}$

$$\mathbf{x}'\left[\mathbf{r}(a') + \beta \sum_{z \in \mathbf{Z}} \gamma(z,a')\right] \geq \mathbf{x}'\left[\mathbf{r}(a) + \beta \sum_{z \in \mathbf{Z}} \gamma(z,a)\right].$$

Then $\gamma' = \mathbf{r}(a') + \beta \sum_{z \in \mathbf{Z}} \gamma(z,a') \in \Psi(\Gamma_{t+1}, \mathbf{X})$ and a' is an optimal action at stage t, given $\mathbf{x}(t) = \mathbf{x}'$.

This result suggests that if we carefully select a finite set of so-called *witness points,* $\{\mathbf{x}^1, \ldots, \mathbf{x}^M\} \subseteq \mathbf{X}$ and use these points to construct corresponding defining vectors $\{\gamma^1, \ldots, \gamma^M\} \subseteq \Psi(\Gamma_{t+1}, \mathbf{X})$, we may produce a good approximation of $\mathbf{PURGE}(\mathbf{G}(\Gamma_{t+1}))$. Ideally, we wish to select a finite set $\mathbf{X}'$ such that $\Psi(\Gamma_{t+1}, \mathbf{X}') = \Psi(\Gamma_{t+1}, \mathbf{X})$. We remark that given this key result, determining $\Psi(\Gamma, \mathbf{X}')$, for finite $\mathbf{X}'$, is straightforward. In this paper, we will use a **GA** to determine such a set of witness points in order to construct a set of γ-vectors that serves as an approximation to the **POMDP** value function.

3. Basics of Genetic Algorithms

Genetic algorithms, introduced in the mid-60's (Fogel, 1962; Fogel et al., 1965; Holland, 1975) aim to solve complex problems by a paradigm from biological evolution. A population of solutions is constructed. A next generation is derived from the old population with operations that promote *survival-of-the-fittest.* Over many generations the solutions in the population tend to improve until the best of the population is (hopefully) near optimal. In the context of the discussion above, a population is a set of witness points.

There are several key components to a **GA** approach: solution encoding, fitness evaluation functions, reproduction processes, and stopping criteria. Instantiations of these fundamental aspects of **GA**s are presented below as we build a **GA** for the **POMDP**. See (Holland, 1975; Davis, 1991; Goldberg, 1989c; Michalewicz, 1996) for additional, general information on **GA**s.

3.1 Encoding Schemes

A *chromosome* is an encoding of a solution and is a vector of variables in $\mathcal{R}^n$. A *gene* is an element of that chromosome vector; an *allele* is a numerical value taken by a gene. In the context of the **POMDP** we propose the following encoding scheme. Recall that we are attempting to find a good set of witness points in X. Let the i^{th} gene of the chromosome denote the probability that the system is in state i. For a 3-state problem, the chromosome is an information state $\mathbf{x} \in \mathcal{R}^3$, x_2, one of its elements, is called a *gene,* and if $x_2 = 0.3$ then the second gene has allele 0.3. Since each chromosome is a distribution, a feasible assignment of alleles must sum to one and each allele must be non-negative. This requirement creates issues with offspring feasibility that will be discussed and resolved in Section 3.3.

3.2 Fitness Evaluation Function

We now determine the real-valued fitness evaluation function (**FEF**) that will be used to measure the fitness of a chromosome.

3.2.1 Selection of the FEF. An **FEF** for the **POMDP** should measure the difference between a value function and its approximation with the intent of improving the quality of the approximation. Given an information state, x, a natural candidate for the **FEF** is

$$f(\mathbf{x}, \Gamma, \Gamma') = \max_{\gamma \in \Gamma} [\mathbf{x}\gamma] - \max_{\gamma \in \Gamma'} [\mathbf{x}\gamma]\,. \tag{15.6}$$

Assume that Γ is the γ-vector set sought, Γ' is its approximation, and both contain no redundant vectors. In results presented later in this paper, we will construct the approximation Γ' so that $\Gamma' \subseteq \Gamma$. Three characteristics of the function $f(\mathbf{x}, \Gamma, \Gamma')$ are:

1 If $\Gamma' = \Gamma$, then $f(\mathbf{x}, \Gamma, \Gamma') = 0$ for all $\mathbf{x} \in \mathbf{X}$.

2 If $\Gamma' \subseteq \Gamma$, then $f(\mathbf{x}, \Gamma, \Gamma') \geq 0$ for all $\mathbf{x} \in \mathbf{X}$.

3 If $\Gamma' \subseteq \Gamma$ and $\Gamma' \neq \Gamma$, then there exists an $\mathbf{x}' \in \mathbf{X}$ such that $f(\mathbf{x}', \Gamma, \Gamma') > 0$.

The function $f(\mathbf{x}, \Gamma_t^{\Psi}, \Gamma_t^{\mathbf{GA}})$ has two limitations. First, all of the information states with 0 fitness value are *indifferent.* They provide no guidance toward identifying peaks for the **GA.** Second, the **GA** has little reason to favor information states close to boundaries or corners in the **PAC** function. Our numerical experience indicates that many missing

γ-vectors are at these corners. To resolve these two issues, we will use the function

$$\mathbf{F}(\mathbf{x}, \Gamma_t^{\Psi}, \Gamma_t^{\mathbf{GA}}) = \left\{ \begin{array}{ll} f(\mathbf{x}, \Gamma_t^{\Psi}, \Gamma_t^{\mathbf{GA}}) & \text{if } f(\mathbf{x}, \Gamma_t^{\Psi}, \Gamma_t^{\mathbf{GA}}) > 0 \\ \Delta(\mathbf{x}, \Gamma_t^{\mathbf{GA}}) & \text{if } f(\mathbf{x}, \Gamma_t^{\Psi}, \Gamma_t^{\mathbf{GA}}) = 0 \end{array} \right\} \quad (15.7)$$

as the **FEF**, where $\Delta(\mathbf{x}, \Gamma_t^{\mathbf{GA}}) = \max^{(2)}_{\gamma \in \Gamma_t^{\mathbf{GA}}} [\mathbf{x}\gamma] - \max_{\gamma \in \Gamma_t^{\mathbf{GA}}} [\mathbf{x}\gamma]$, and $\max^{(2)}_{\gamma \in \Gamma_t^{\mathbf{GA}}} [\mathbf{x}\gamma]$ is the second-best value from $\Gamma_t^{\mathbf{GA}}$ when all γ-vectors are evaluated at $\mathbf{x}$.

We motivate our interest in using $\mathbf{F}(\mathbf{x}, \Gamma_t^{\Psi}, \Gamma_t^{\mathbf{GA}})$ as the **FEF** as follows. If $f(\mathbf{x}', \Gamma_t^{\Psi}, \Gamma_t^{\mathbf{GA}}) > 0$, then a new defining γ-vector can be found, so set the fitness of the witness point $\mathbf{x}'$ to be equal to $f(\mathbf{x}', \Gamma_t^{\Psi}, \Gamma_t^{\mathbf{GA}})$. If $f(\mathbf{x}, \Gamma_t^{\Psi}, \Gamma_t^{\mathbf{GA}}) = 0$, this information state finds an existing vector in $\Gamma_t^{\mathbf{GA}}$, so set the fitness of such an information state to be equal to the difference between the second-best and the best function value that $\Gamma_t^{\mathbf{GA}}$ can attain at information state $\mathbf{x}$. This function can help the **GA** differentiate information states and can drive the population toward the undiscovered regions. These regions are most likely found at corners or boundaries.

3.2.2 Characteristics of $\mathbf{F}(\mathbf{x}, \Gamma_t^{\Psi}, \Gamma_t^{\mathbf{GA}})$. Use of $\mathbf{F}(\mathbf{x}, \Gamma_t^{\Psi}, \Gamma_t^{\mathbf{GA}})$ as the **FEF** generates two types of local maxima: those with fitness value less than or equal to 0 and those with fitness value strictly greater than 0. The first type will not give new defining vectors and are referred to as *fruitless* attractors. The second type gives new defining vector(s) and are referred to as *fruitful* attractors. In general, there are two characteristics of $\mathbf{F}(\mathbf{x}, \Gamma_t^{\Psi}, \Gamma_t^{\mathbf{GA}})$: a) the number of attractors can grow large, and b) the number of fruitless attractors may outnumber fruitful attractors. Thus, we hypothesize that the degree of difficulty faced by the **GA** increases as the number of defining vectors identified increases.

This hypothesis, in fact, touches the core issue of what makes a problem difficult for a **GA**. Several criteria, such as isolation, deception, and multimodality (Goldberg, 1993), have been suggested as measures of a problem's level of difficulty for a **GA**. See (Goldberg, 1989a; Goldberg, 1989b; Goldberg, 1991; Goldberg et al., 1992; Horn and Goldberg, 1995; Whitley, 1991) for further discussion. The function $f(\mathbf{x}, \Gamma_t^{\Psi}, \Gamma_t^{\mathbf{GA}})$ has a high degree of isolation where fruitful attractors are closely surrounded by points with low fitness value. The isolation level of the **FEF** $\mathbf{F}(\mathbf{x}, \Gamma_t^{\Psi}, \Gamma_t^{\mathbf{GA}})$ is low, while its level of deception (fruitless attractors) and multimodality (more than one attractor) are both higher. Despite these difficulties, preliminary numerical results reported in (Lin, 1999) suggest that the use of $\mathbf{F}(\mathbf{x}, \Gamma_t^{\Psi}, \Gamma_t^{\mathbf{GA}})$ will most likely yield a better ap-

proximation to a solution. To counter the difficulties in $\mathbf{F}(\mathbf{x}, \Gamma_t^{\Psi}, \Gamma_t^{\mathbf{GA}})$ we will exploit certain problem-domain knowledge in a partially guided search scheme presented in Section 4.2.1.

3.3 Reproduction Processes

The three common operations in **GA**s are selection, crossover, and mutation. Selection determines which chromosomes will bear offspring. Typically, chromosomes with higher fitness are preferred (Holland, 1975). Crossover performs the transformation of parents into offspring. Mutation adds randomness to the process, making it ergodic.

In selecting chromosomes for breeding the next generation, we pool chromosomes in the old population together with the set of witnesses corresponding to the γ-vectors in the current $\Gamma_t^{\mathbf{GA}}$. For selection, the basic idea is to pair two distinct chromosomes hitting different identified defining vectors. This tends to reduce the occurrence of fruitless points that arise when both parents witness the same defining vector.

Typical crossover operators include one-point crossover, two-point crossover (Cavicchio, Jr., 1970; Goldberg, 1989c), and uniform crossover (Spears and De Jong, 1991; Syswerda, 1989). We now illustrate one-point crossover for the **POMDP** information state encoding. Recall that a chromosome in our encoding is an information state and that an information state is a probability distribution. Two such information states for a 3-state problem are (.2, .2, .6) and (.4, .4, .2). The chromosomal representation of such information states is simply the probability vector itself. A one-point crossover would cleave each chromosome at some point, say after the first gene, and exchange leading segments. Performing that operation over the two example chromosomes gives two offspring (.2, .4, .2) and (.4, .2, .6). Neither sum to 1 as is necessary for a probability mass vector. See (Michalewicz, 1996) for remedial methods to deal with this offspring infeasibility problem. An obvious fix is to normalize by dividing each allele by the sum of alleles for that chromosome. For example, (.2, .4, .2) and (.4, .2, .6) become $(\frac{1}{4}, \frac{1}{2}, \frac{1}{4})$ and $(\frac{1}{3}, \frac{1}{6}, \frac{1}{2})$. Note that multiple chromosomes will map to the same probability mass vector. This could confound the GA, though we saw no evidence of that in our experiments. Alternatively, specialized crossover operators (e.g., linear convex combination), repair-mechanisms (e.g., normalization), constraint handling techniques (Michalewicz, 1996), or random keys (Bean, 1994) can be used to ensure the feasibility of offspring. We remark that preliminary numerical experiments in (Lin, 1999) show that random keys performs better than the other schemes in terms of solution quality yielded and number of defining vectors identified. Hence,

we adopt the random keys approach in the **GA**s presented later in this chapter.

The use of random keys places no restriction on crossover operators. To probabilistically enhance the diversity of the population, we propose a *composite crossover operator* over two chosen parents to create two offspring. This operator, which involves uniform crossover and linear combination, is detailed in the appendix.

We adopt two types of mutation: immigration and the Move_to_Boundary operation. Immigration randomly generates a few new points to be included in each new population. Their genetic material is mixed through the population in subsequent generations. Move_to_Boundary randomly selects a gene of a chromosome and forces the allele of that gene to become 0, bringing that chromosome to the boundary corresponding to the selected gene. This re-expands the search by countering the tendency to regress to the mean.

3.4 Stopping Conditions

To determine the stopping time in the **GA** search, three possible criteria are used in combination: no new defining vectors found in several consecutive generations, a generation threshold on the number of generations, and return rate of the **GA** (the ratio of the number of newly found defining vectors in a single run of **GA** to its population size). Use of these criteria is described in Section 4.

4. Proposed Genetic Algorithm Heuristic

In this section, we present a **GA**-based heuristic for determining a finite witness point set $\mathcal{W}_t$ for the operation $\Gamma_t^{\mathbf{GA}} = \Psi(\Gamma_{t+1}^{\mathbf{GA}}, \mathcal{W}_t)$. This heuristic, called DC-NICHE, is discussed in Section 4.2. Section 4.1 presents random keys, the foundation of the heuristic. Section 4.4 presents complexity reduction procedures.

4.1 Random Keys

Random keys (Bean, 1994) is a robust yet simple method to ensure the feasibility of offspring without disrupting the search. Successful applications can be found in (Bean, 1994; Hadj-Alouane et al., 1999; Norman and Bean, 2000; Norman and Bean, 1999; Norman and Bean, 1997). It is very similar to the Argot strategy in (Shaefer and Smith, 1988). Note that the literature is limited to discrete solution spaces. In this paper we extend the use of random keys to the information space, a continuous space.

The random keys **GA** works with two spaces: search space and literal space. In genetic terminology, the random keys are a genotype and the normalized information vector is a phenotype. All of the **GA** operators act upon the search space, the genotypes. Then each individual chromosome is transformed into a feasible solution and evaluated in the literal space. Figure 15.1 (a) depicts the random keys processes. The random keys representation encodes a solution with random numbers. A mapping decodes the solution. The search space typically is a unit hypercube in $\mathcal{R}^n$. For this problem, the mapping function to decode the solution is normalization. Figure 15.1 (b) gives an example of a 3-state problem. The random keys approach differs from the simple normalization in that the unnormalized random keys are maintained. The normalization is used to obtain a fitness value rather than to alter the chromosome. This difference is crucial in that it reduces disruption of the search.

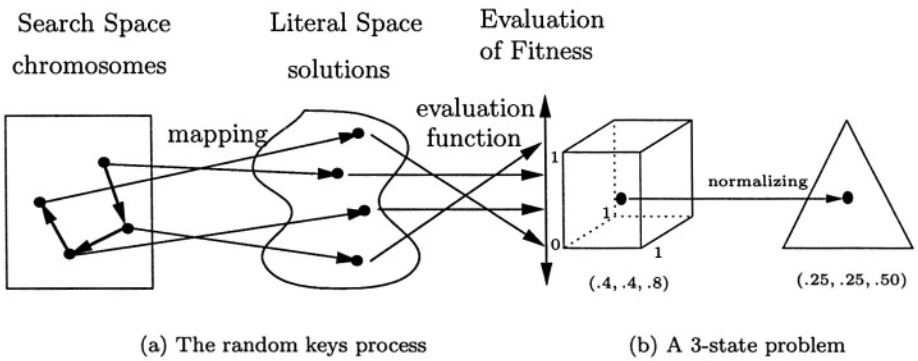


Figure 15.1. The random keys process and an example for a 3-state problem

Recall that $\mathcal{W}_t$ represents the set of witness points used to construct the γ-vector set $\Gamma_t^{\mathbf{GA}}$; i.e., $\Gamma_t^{\mathbf{GA}} = \Psi(\Gamma_{t+1}^{\mathbf{GA}}, \mathcal{W}_t)$. We assume $\mathcal{W}_t = \mathcal{W}_t^{new} \cup \mathcal{W}_{t+1}$. The algorithm *Basic GA* summarizes the pseudocode of the basic genetic algorithm (**GA**) for the finite horizon **POMDP**, and Table 15.1 provides definitions of the notation used.

There are two loops in *Basic GA*. The outer loop is for the problem horizon to be solved, (t, T). The inner loop is the core of **GA**. **Step 1.0** initializes parameters. **Step 1.1** sets up an initial population for the **GA**. The members of this initial population come from three sources: the set of old witnesses, $\mathcal{W}_{t+1}$, extreme points of the information space, and randomly generated points. This step also constructs an initial Γ-set for the value function at stage t, $\Gamma_t^{\mathbf{GA}}$. For the purpose of computational performance, the γ-vectors in $\Gamma_t^{\mathbf{GA}}$ are uniquely numbered in the order

n	=	the generation counter for the **GA**.
P_t^n	=	the population for the n^{th} generation at stage t.
N_t	=	the size of the population at stage t.
N^*	=	the minimal size of the population.
b	=	a constant multiplier.
$\mathcal{W}_t$	=	a set of witnesses for stages t to T, each of them termed as an old witness.
$\mathcal{W}_t^{new}$	=	a set of witnesses associated with the Γ-set for stage t.
$\bar{\mathcal{W}}$	=	a set of witnesses identified only at n^{th} generation in the reproduction processes for stage t.
$\mathcal{B}_t^n$	=	the pool of parents selected to breed the next population P_t^{n+1}
M_{r_rate}	=	the minimal threshold of yield rate of the new defining vector in a generation, > 0.
r_rate	=	the yield rate of new defining vectors − fraction of new points witnessing new vectors.
M_{0_run}	=	the maximal consecutive generations yielding *insufficient* ($r\_rate < M_{r\_rate}$) new defining vectors, ≥ 1
0_run	=	the counter for tracking the number of generations that yield *insufficient* defining vectors in a row.
M_{g_limit}	=	the maximal generation limit, ≥ 1.

Table 15.1. Definitions of parameters used in the basic **GA**

they are identified. **Step 1.2** checks the termination criteria. **Step 1.3**, the reproduction process, is the core of the **GA** and drives the initial population toward desirable solutions. There are four sub-steps here. **Step 1.3.1** focuses the search on corners. **Step 1.3.2** selects the pool of parents. **Step 1.3.3** is the crossover operation *between two randomly selected parents from* $\mathcal{B}_t^n$, and is purposely left unspecified. The next section of the paper proposes the strategy DC-NICHE to accomplish this step. In **Step 1.3.4** Partial_Guided operator exploits the structure of the evaluation function defined in equation (15.7), and will be discussed later. Move_to_Boundary and immigration operations are the mutation operators. Note that t, T, Γ_t, and $\Gamma_t^{\mathbf{GA}}$ are defined as before, except that Γ_t is approximated by the solution set obtained by $\Gamma_t^{\mathbf{GA}}$. The notation $x \leftarrow y$, $x++$, and $x--$ denote to overwrite x with y, increment x by 1, and decrement x by 1, respectively.

Basic GA {
Step 0. Set $t \leftarrow T-1$, $\Gamma_{T-1}^{\mathbf{GA}} \leftarrow \emptyset$, $\mathcal{X}_T \leftarrow \emptyset$, $\Gamma_T \leftarrow \{\bar{\mathbf{r}}\}$.
Step 1. Stop the **GA** if $t < 0$. Otherwise continue.
Step 1.0. Set $n \leftarrow 0$, $\mathcal{X}_t^{new} \leftarrow \emptyset$, $N^* \leftarrow 100$, $b \leftarrow 3$, and $N_t \leftarrow \max\{N^*, |\Gamma_{t+1}| + (T-t)b\}$. Let $0\_run \leftarrow -1$.
Step 1.1. Initialize a population P_t^n with size N_t consisting of the following three groups:
Old witnesses: If $\mathcal{X}_{t+1} \neq \emptyset$, evaluate and include old witnesses yielding new defining vectors to P_t^n as well as to

$\mathcal{X}_t^{new}$, and the associated defining vectors to $\Gamma_t^{\mathbf{GA}}$.

Extreme points: If$|\mathcal{X}_t^{new}| < N_t$, evaluate and include extreme points of $\mathbf{X}$ yielding new defining vectors to P_t^n as well as to $\mathcal{X}_t^{new}$, and the associated defining vectors to $\Gamma_t^{\mathbf{GA}}$. Extreme points of $\mathbf{X}$ are $e_i, i \in \mathbf{S}$, whose i^{th} entry is 1; 0 elsewhere.

Random points: If $|\mathcal{W}_t^{new}| < N_t$, evaluate and include $(N_t - |\mathcal{W}_t^{new}|)$ randomly generated points to P_t^n. If any of them yields new defining vectors, include it to $\mathcal{W}_t^{new}$, and the associated defining vector to $\Gamma_t^{\mathbf{GA}}$.

Modify N_t: $N_t \leftarrow \max \{|\mathcal{W}_t^{new}|, N_t\}$.

Step 1.2. Stop and go to **Step 2** if a) no new defining vectors are found in $0\_run \geq M_{0\_run}$ generations in a row, or b) $n \geq M_{g\_limit}$. Continue otherwise.

Step 1.3 Reproduction processes

Step 1.3.1 Negate the fitness of new witnesses in $\mathcal{W}_t^{new}$ if positive, and set $\bar{\mathcal{W}} \leftarrow \emptyset$.

Step 1.3.2 The pool of parents B_t^n to breed a new population P_t^{n+1} consists of all witnesses in $\mathcal{W}_t^{new}$ and chromosomes in P_t^n. Hence, $B_t^n \leftarrow (\mathcal{W}_t^{new} \cup P_t^n)$.

Step 1.3.3 Perform crossover between randomly selected parents from B_t^n and include new witnesses into the set $\bar{\mathcal{W}}$ and the associated defining vectors to $\Gamma_t^{\mathbf{GA}}$.

Step 1.3.4 Perform Partial_Guided, Move_to_Boundary, and immigration operation and include any new witnesses to the set $\bar{\mathcal{W}}$ and the associated defining vectors to $\Gamma_t^{\mathbf{GA}}$.

Step 1.4 Compute $r\_rate \leftarrow \frac{|\bar{\mathcal{W}}|}{N_t}$. If $r\_rate < M_{r\_rate}$, 0_run++; otherwise $0\_run \leftarrow 0$. Set $P_t^{n+1} \leftarrow P_t^n$, $\mathcal{W}_t^{new} \leftarrow \mathcal{X}_t^{new} \cup \bar{\mathcal{W}}$, $n++$, and go to **Step 1.2**.

Step 2. Set $\mathcal{W}_t \leftarrow \mathcal{W}_{t+1} \cup \mathcal{W}_t^{new}$, $\Gamma_t \leftarrow \Gamma_t^{\mathbf{GA}}$, $t--$, and go to **Step 1.**

}

4.2 Stratifying the GA Search: DC-NICHE

In this section, we present our **GA**-based procedure, DC-NICHE, for determining the witness point sets $\mathcal{W}_t$, $t = 0, 1, \ldots, T-1$. The **FEF** defined in Section 3 is typically multimodal. If an attractor is fruitful (fruitless), it is defined by previously unidentified (identified) γ-vector(s). A typical **GA** will search for the global optimum of this function. However, we are interested in all attractors, especially fruitful

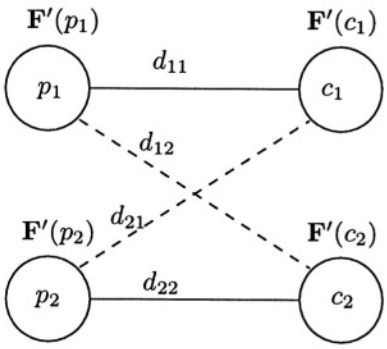
attractors, to determine undiscovered γ-vectors. To stratify the search we employ *niche* approaches from the **GA** literature.

There are several niche schemes related in the **GA** literature, including crowding (De Jong, 1975; Mahfoud, 1995), spatial selection (Gorges-Schleuter, 1990; Davidor, 1991), and fitness sharing (Holland, 1975; Goldberg and Richardson, 1987; Holland, 1992). Niching corresponds to a natural phenomenon that similar species, in general, will compete for the limited resources available in the same living niche. Niching methods are techniques to promote the formation and maintenance of several stable subpopulations by promoting population diversity (Mahfoud, 1995). Each subpopulation is expected to give *near-local-optimum* solutions. Therefore, a niching **GA** is expected to produce multiple distinct solutions in a single run. For our nonlinear function, we hope to find multiple fruitful attractors and, hence, multiple new γ-vectors.

The *deterministic-crowding niche* algorithm (DC-NICHE) uses crowding, a specific selection scheme that promotes diversity. The basis of crowding is a method to measure distance between chromosomes. Theory and methods of measuring the similarity between two elements can be found in (Goldberg, 1989c; Holland, 1992). For the purpose of computational speed, the maximum componentwise deviation between two elements is employed. In particular, this algorithm is modified from deterministic crowding in (Mahfoud, 1995). When two parents bear two offspring, we choose to keep the parents or to replace them with the offspring based on the closeness of the parents to the offspring. Further, which parent is replaced by which offspring is selected to intensify niche formation.

The basic mechanism of pairing up parents and children employed in this modified crowding scheme is demonstrated in Figure 15.2 (Mahfoud, 1995). Let p_1, p_2 be two randomly chosen parents and c_1, c_2 be their offspring. We will discuss how the parents are chosen later. Let d_{ij} be the maximum componentwise deviation between p_i and c_j, and let $\mathbf{F}'(p_i)$ be the fitness of parent p_i, $i, j = 1, 2$. The exact scheme is given in the pseudocode in Figure 15.2.

To implement DC-NICHE replace **Step 1.3.3** in the algorithm of *Basic GA* with the pseudocode sketched as the following, termed as **Step 1.3.3**$^{DC-NICHE}$. This pseudocode details the crossover mechanism for the DC-NICHE algorithm, where the composite crossover operator is defined in the appendix. We remark that its **Step 1.3.3.1,** not only reduces the occurrence of fruitless points that arise when both parents witness the same defining vector, but probabilistically favors the defining vectors with larger defining regions. **Step 1.3.3.2** increases the



if($d_{11} + d_{22} \leq d_{12} + d_{21}$)
  $p_1 \leftarrow c_1$ if($\mathbf{F}'(p_1) < \mathbf{F}'(c_1)$)
  $p_2 \leftarrow c_2$ if($\mathbf{F}'(p_2) < \mathbf{F}'(c_2)$)
else
  $p_1 \leftarrow c_2$ if($\mathbf{F}'(p_1) < \mathbf{F}'(c_2)$)
  $p_2 \leftarrow c_1$ if($\mathbf{F}'(p_2) < \mathbf{F}'(c_1)$)
end

Figure 15.2. Illustration of crowding scheme

probability of witnesses associated with defining vectors having smaller defining regions to be selected as parents. Note that **Step 1.3.3.1.1** defines the selection method on the current population without replacement, while the selection method on a set of witnesses for **Step 1.3.3.2** is carried out with replacement.

Step 1.3.3$^{DC-NICHE}$ **Crossover operation step for the DC-NICHE algorithm {**

Step 1.3.3.1 Let k' be the number of distinct γ-vectors following the last generation. Let h be the total number of chromosomes contributed by Move_to_Boundary, immigration, and Partial_Guided operations in the new population. Each defining vector is uniquely numbered. Define $k = \min\{\frac{k'}{2}, \lfloor\frac{N_t - h}{2}\rfloor\}$. Then perform the following steps k times:

Step 1.3.3.1.1 Without replacement, randomly select two distinct parents, p_1 and p_2, hitting different identified defining vectors.

Step 1.3.3.1.2 Perform a composite crossover on p_1 and p_2 to create two offspring, c_1 and c_2.

Step 1.3.3.1.3 Evaluate the two newly bred offspring.

Step 1.3.3.1.4 Execute the modified crowding scheme illustrated in Figure 15.2 to pair up parents and offspring and meanwhile to determine who should survive into the next generation for each parent-child pair.

Step 1.3.3.2 Define $m = \max\{0, \lfloor\frac{N_t - 2*k - h}{2}\rfloor\}$. Performing m crossover steps as described in **Step 1.3.3.1**, except

a) both parents are randomly selected from the set $\mathcal{W}_t^{new}$, instead of from P_t^n, and b) parents are selected with replacement.

}

4.2.1 Partially Guided Search. We can enhance DC-NICHE with *partially guided* search. This search scheme is not time consuming and slightly improves the solution discovered. Each $\gamma^1 \in \Gamma_t^{\mathbf{GA}}$ is associated with a witness, $\bar{\mathbf{x}}$. Let $\gamma^2 \in \Gamma_t^{\mathbf{GA}}$ be the second best vector in $\Gamma_t^{\mathbf{GA}}$ when evaluated at $\bar{\mathbf{x}}$. These vectors cross at

$$\{\mathbf{x} \in \mathbf{X} : (\gamma^1 - \gamma^2)\mathbf{x} = 0\}. \tag{15.8}$$

If one of these crossing points, $\mathbf{x}^*$, witnesses a new defining vector, then add the vector to $\Gamma_t^{\mathbf{GA}}$. If not, then move $\bar{\mathbf{x}}$ along the direction toward $\mathbf{x}^*$ some random amount. Then the updated $\bar{\mathbf{x}}$ remains a witness, but moves closer to a corner or boundary.

4.3 Complexity Reduction Procedures

The complexity of the proposed **GA** is $O(\Upsilon|\mathbf{A}||\mathbf{Z}||\mathbf{S}||\Gamma_{t+1}^{\mathbf{GA}}|)$, where Υ is the total number of points in the population when **GA** stops running. Thus, if a defining vector is discarded from $\Gamma_{t+1}^{\mathbf{GA}}$, then the computation will be reduced by $O(\Upsilon|\mathbf{A}||\mathbf{Z}||\mathbf{S}|)$. The actual reduction can be magnified if many defining vectors are discarded from $\Gamma_{t+1}^{\mathbf{GA}}$ because Υ often decreases when $\Gamma_{t+1}^{\mathbf{GA}}$ gets smaller. This observation motivates discarding some defining vectors that may contribute negligibly to the function value. We refer to this as the Γ-cut mechanism, which is motivated from the approximation schemes in (Cassandra et al., 1997).

If a newly discovered defining vector alters the fitness function by a very small amount, we will discard it at the risk of loss of optimality. The impact of this lost vector may propagate to earlier stages. An analysis of error caused by this propagation is presented in Section 5.

Let ϵ be the fitness threshold for including a new defining vector, γ. Let Γ be the set of known defining vectors prior to discovering γ. Our theoretical objective is to discard γ if $\sup_{\mathbf{x}\in\mathbf{X}} \mathbf{F}(\mathbf{x}, \Gamma \cup \{\gamma\}, \Gamma) < \epsilon$. This is time-consuming to evaluate in practice. In our **GA** heuristics, a new defining vector identified by $\mathbf{x}$ will not be included into Γ if $\mathbf{F}(\mathbf{x}, \Gamma \cup \{\gamma\}, \Gamma) < \epsilon$. This may lead to greater error than ϵ as the initial error propogates through the recursion.

We can refine this process by recognizing that defining vectors identified early in the process have a greater impact on propagated error. We

then use lower ϵ values if the cardinality of Γ is small, and increase ϵ as $|\Gamma|$ increases.

In Lin (1999), two examples of threshold functions, $\mathcal{C}_c$ and $\mathcal{C}_v$, were proposed and numerically investigated. Fitness threshold function $\mathcal{C}_c$ is a constant function with very small positive value, $\epsilon > 0$. Fitness threshold function $\mathcal{C}_v$ is a piecewise linear function and is bounded above by 0.005 if $|\Gamma_t^{\mathbf{GA}}| \geq 400$. Its precise definition is presented in Table A.3 in the appendix. Preliminary numerical results show that, in general, the proposed **GA**s with the use of fitness threshold function $\mathcal{C}_v$ yield good approximations and are computationally tractable. Thus, we will adopt this fitness threshold function in Section 6.

5. Heuristic Performance Measures

This section discusses three measures to study the performance of several algorithms for the **POMDP**.

5.1 Error Upper Bound, η

The error upper bound between two **PAC** value functions represented by Γ_t and $\Gamma_t^{\mathbf{GA}}$ is defined as $\eta_t = \max_{\mathbf{x}\in\mathbf{X}} f(\mathbf{x}, \Gamma_t, \Gamma_t^{\mathbf{GA}})$. This 'a posteriori' error upper bound will be computed using a mixed integer program (**MIP**) to evaluate the quality of the solutions produced by the heuristics. The exact Γ-sets, Γ_t's, are obtained through use of the most efficient optimal algorithm thus far reported (Cassandra et al., 1997), which is referred to as *lp-best.*

Let H be a sufficiently large positive number and let the 0 –1 variable y_γ correspond to vector γ in Γ_t. The $v_t(\mathbf{x})$ is defined as before and $z_t(\mathbf{x})$ is its approximation yielded by the **GA**. The following **MIP**, having as its objective to maximize the difference of two **PAC** value functions, $[v_t(\mathbf{x}) - z_t(\mathbf{x})] \equiv f(\mathbf{x}, \Gamma_t, \Gamma_t^{\mathbf{GA}})$, yields the error upper bound, η_t, and is defined as follows:

$$
\begin{array}{rll}
\eta_t \equiv & \text{MAX} & v_t(\mathbf{x}) - z_t(\mathbf{x}) \\
& \text{S.T.} & v_t(\mathbf{x}) \leq \mathbf{x}\gamma + (1 - y_\gamma)H, \text{ for all } \gamma \in \Gamma_t, \\
& & \sum_{\gamma\in\Gamma_t} y_\gamma = 1, \\
& & y_\gamma \in \{0, 1\}, \text{ for all } \gamma \in \Gamma_t, \\
& & z_t(\mathbf{x}) \geq \mathbf{x}\gamma, \text{ for all } \gamma \in \Gamma_t^{\mathbf{GA}}, \\
& & \mathbf{x} \in \mathbf{X}.
\end{array}
\tag{15.9}
$$

The value of η_t signifies the maximum error caused by the heuristic solution at stage t. If $\eta_t = 0$, then the heuristic solution is in fact optimal. If $\eta_t > 0$, then the difference between the heuristic **PAC** curve and the optimal **PAC** curve is no greater than η_t.

However, for 1p-best the **POMDP** is intractable when problem sizes grow. Thus, it becomes infeasible to compute η_t directly. Alternatively, we will compute the error estimate caused by our heuristic solutions, η_t^ϵ, relative to an approximation constructed by the 1p-best with the ϵ-Γ-cut operation (denoted as *lp-best*(ϵ)). Let the approximate solution set obtained with lp-best(ϵ) be denoted as Γ_t^ϵ.

An ϵ-Γ-cut prunes defining vectors with function value improvement less than $\epsilon > 0$ much as in Section 4.3, though applied to lp-best. Assuming ϵ is used throughout, lp-best(ϵ) recursively uses this ϵ-Γ-cut step at most $(2 \times |\mathbf{A}||\mathbf{Z}| - 1)$ times in order to construct Γ_t^ϵ. The error of the lp-best(ϵ), $\eta_t^{lp-best(\epsilon)}$, is the accumulation of errors attributed to the implementation of lp-best(ϵ) to solve (15.3-15.5):

We remark, based upon our numerical results reported later, that the approximate **PAC** curve constructed by lp-best(ϵ) could be entirely below the optimal **PAC** curve, especially when $|\mathbf{Z}|$ gets large.

When we substitute Γ_t^ϵ for Γ_t, this **MIP** yields the error estimation, η_t^ϵ, associated with the heuristic solution; i.e., $\eta_t^\epsilon = \max_{\mathbf{x} \in \mathbf{X}} f(\mathbf{x}, \Gamma_t^\epsilon, \Gamma_t^{\mathbf{GA}})$. Note that η_t^ϵ can be negative. If $\eta_t^\epsilon < 0$, then the entire heuristic **PAC** curve is above that of lp-best(ϵ) by at least $-\eta_t^\epsilon$. If $\eta_t^\epsilon > 0$, then the difference between our heuristic **PAC** curve and the lp-best(ϵ) **PAC** curve is no greater than η_t^ϵ. However, it is possible that portions of our heuristic **PAC** curve may give better function values than that of lp-best(ϵ).

The relation among η_t, η_t^ϵ, and $\eta_t^{lp-best(\epsilon)}$ $(\equiv \max_{\mathbf{x} \in \mathbf{X}} f(\mathbf{x}, \Gamma_t, \Gamma_t^\epsilon))$ can be characterized as follows:

$$\begin{aligned} 0 \leq \eta_t &= \max_{\mathbf{x} \in \mathbf{X}} f(\mathbf{x}, \Gamma_t, \Gamma_t^{\mathbf{GA}}) \\ &= \max_{\mathbf{x} \in \mathbf{X}} \left\{ f(\mathbf{x}, \Gamma_t^\epsilon, \Gamma_t^{\mathbf{GA}}) + f(\mathbf{x}, \Gamma_t, \Gamma_t^\epsilon) \right\} \\ &\leq \eta_t^\epsilon + \eta_t^{lp-best(\epsilon)}. \end{aligned}$$

This suggests a way to compute a looser error upper bound of η_t, $\eta_t^\epsilon + \eta_t^{lp-best(\epsilon)}$, if it is infeasible to obtain η_t directly. The derivation of such a bound, $\eta_t^{lp-best(\epsilon)}$, is a topic for future research. We will only report the results of η_t^ϵ's in Section 6.

5.2 Error Ratio Upper Bound, ρ

The *error ratio upper bound*, $\rho_t \equiv \max_{\mathbf{x} \in \mathbf{X}} \frac{v_t(\mathbf{x}) - z_t(\mathbf{x})}{v_t(\mathbf{x})}$, is the maximal error ratio of the heuristic **PAC** curve relative to the optimal **PAC** curve. Considering a pair of defining vectors $\gamma^v \in \Gamma_t$ and $\gamma^z \in \Gamma_t^{\mathbf{GA}}$ at a time, we obtain this value ρ_t from solving $|\Gamma_t||\Gamma_t^{\mathbf{GA}}|$ such nonlinear

models as:

$$\begin{array}{rll} \rho_t^{v,z} \equiv & \text{MAX} & \frac{\mathbf{x}(\gamma^v - \gamma^z)}{\mathbf{x}\gamma^v} \\ & \text{S.T.} & \mathbf{x}\gamma^v \geq \mathbf{x}\gamma, \text{ for all } \gamma \in \Gamma_t; \\ & & \mathbf{x}\gamma^z \geq \mathbf{x}\gamma, \text{ for all } \gamma \in \Gamma_t^{\mathbf{GA}}; \\ & & \mathbf{x} \in \mathbf{X}. \end{array} \tag{15.10}$$

Each such model computes the *local maximum error ratio,* $\rho_t^{v,z}$, over the intersection of the defining regions of γ^v and γ^z, $\mathcal{D}_{\mathbf{X}}(\gamma^v, \gamma^z)$, as defined by the set of constraints in (15.10). Thus,

$$\rho_t = \max_{\gamma^v \in \Gamma_t,\ \gamma^z \in \Gamma_t^{\mathbf{GA}}} \rho_t^{v,z}.$$

A pair of γ^v and γ^z is referred to as a *feasible pair* if $\mathcal{D}_{\mathbf{X}}(\gamma^v, \gamma^z) \neq \emptyset$. We remark that assuming $\gamma^v \mathbf{x} > 0$ (as it is in the examples tested in this paper), and γ^v and γ^z is a feasible pair, the implications of the sign of $\rho_t^{v,z}$ is the same as that of η_t. However, the region considered now is $\mathcal{D}_{\mathbf{X}}(\gamma^v, \gamma^z)$, not X. Note that ρ_t^{ϵ} is analogously defined.

In fact, (15.10) is a linear fractional program, whose solution procedure has been established in (Charnes and Cooper, 1962; Murty, 1983). The theory and procedure employing two **LP**s to solve (15.10) is detailed in (Murty, 1983). Technical detail is presented in (Lin, 1999).

5.3 Total Used Points, Υ

To obtain $\Gamma_0^{\mathbf{GA}}$ for approximating the value function at the current stage requires determination of all other Γ-sets from stage 1 to stage $T-1$. At stage t, our proposed **GA**s will generate many points, m_t $(\geq |\mathcal{W}_t^{new}|)$, to construct an approximation of the value function sought. Ideally, we would like to use as few points $(\Upsilon = \sum_{t=0}^{T-1} m_t)$ as possible while constructing a good approximation. Moreover, based upon numerical experiences, we found that for $t = 0, 1, \ldots, T-1$, the number, m_t, helps to define the design parameter d_t at stage t for the fixed grid based method in (Lovejoy, 1991). This is justified in Section 6 by the high quality solution yielded by the *informed fixed grid method* (**IFG**).

The **IFG** is a modified procedure of the fixed grid method in (Lovejoy, 1991). In (Lovejoy, 1991), the grid of points is generated by triangulating the original simplex **X** with the design parameter d, divisions over each face of **X**. The value of d is closely related to the quality of approximation derived from the corresponding grid of points. As indicated in (Lovejoy, 1991), the number of points will explode for high values of $|\mathbf{S}|$ and d, and there are no a priori bounds available to aid in choosing the grid

appropriate for a desired solution quality. Rather, the **IFG** method first decides a d_t-value to define a grid of points $\mathbf{X}_t^{\mathbf{IFG}}$ such that $|\mathbf{X}_t^{\mathbf{IFG}}|$ is as close to m_t as possible. Given the set $\mathbf{X}_t^{\mathbf{IFG}}$ and $\Gamma_{t+1}^{\mathbf{IFG}}$, determining the approximate **PAC** curve of the **IFG** method, $\Psi(\Gamma_{t+1}^{\mathbf{IFG}}, \mathbf{X}_t^{\mathbf{IFG}})$, is straightforward.

6. Numerical Results

This section reports tests on performance of three approximation methods: DC-NICHE (presented here), informed-fixed-grid (**IFG**) (Lovejoy, 1991), and lp-best(ϵ) (Cassandra et al., 1997).

Experiments were run on 2 examples from (Cheng, 1988) and 8 randomly generated examples. These 10 examples are briefly described in Table A.1 in the appendix. We assume a horizon of $T = 20$. Parameter values for DC-NICHE are detailed in Table A.2. Note that the fitness threshold function $\mathcal{C}_v$ is also used to screen points in the **IFG** approach and thus the defining vectors they identify. The solution obtained by the lp-best(ϵ) method with $\epsilon = .01$ is the reference point of comparison. That is, all errors are reported by deviation from the value obtained by the lp-best(ϵ=.01).

The criteria to evaluate the performance of an algorithm are the error upper bound (η), the error ratio upper bound (ρ), the total run-time ($\mathcal{T}$), and the total number of points sampled (Υ). In general, we favor an algorithm with small η, small ρ, small $\mathcal{T}$, and small Υ.

Tables 15.2, 15.3, 15.4, and 15.5 summarize the numerical results. Note that, $\eta_{0,20}^{\epsilon}$ is an error estimation that the corresponding algorithm can produce relative to the solution of the lp-best(ϵ) at stage 0. We find this by solving the corresponding **MIP** model in (15.9). The other criteria, $\rho_{0,20}^{\epsilon}$, $\mathcal{T}_{0,20}$, and $\Upsilon_{0,20}$ are defined and interpreted in a similar manner. A negative value of $\eta_{0,20}^{\epsilon}$ signifies that the corresponding algorithm obtains a better approximation than the solution of the lp-best(ϵ).

Table 15.2 shows the solution quality for the four approaches. For the small problems (p444, p533) DC-NICHE and **IFG** produce virtually equivalent solutions that are better than lp-best(ϵ=.01). For the problems with large state spaces (p2044, p3044, p5044, p10044), lp-best(ϵ=.01) is superior for at least one information state for each problem. For the problems with large observation spaces (p4204, p4034, p4504, p41004) DC-NICHE and **IFG** consistently beat lp-best(ϵ=.01). Algorithm DC-NICHE is generally best over all classes of problems.

Recall from Section 5.2 that the relative percent error value, ρ^{ϵ}, is computed in a manner biased toward lp-best(ϵ). For each feasible pair of defining vectors with joint defining region $\mathcal{D}$, lp-best(ϵ) is declared the

Table 15.2. Comparisons among various algorithms: solution quality

strategy	DC-NICHE		IFG	
examples	$\eta^{\epsilon}_{0,20}$	$\rho^{\epsilon}_{0,20}$	$\eta^{\epsilon}_{0,20}$	$\rho^{\epsilon}_{0,20}$
p444	-.016513	-.000104	-.016513	-.000104
p533	-.000788	-.000005	-.000788	-.000005
p2044	.013921	.001527	.017160	.001852
p3044	.032403	.006549	.035069	.008731
p5044	.081604	.016302	.094347	.023442
p10044	.042361	.031873	.042361	.033579
p4204	-.089557	-.006800	-.089527	-.006784
p4304	-.429670	-.080633	-.423378	-.080619
p4504	-.351439	-.006524	-.346330	-.006530
p41004	-.392682	-.049614	-.390409	-.049521

Table 15.3. Comparisons among various algorithms: distribution of feasible-pair-win-percentage vs. lp-best(ϵ), %

problem	method	feasible pairs	win %
p2044	DC-NICHE	89	91.01
	IFG	88	93.18
p3044	DC-NICHE	2150	86.70
	IFG	2216	90.88
p5044	DC-NICHE	89167	49.50
	IFG	81105	38.58
p10044	DC-NICHE	68248	89.06
	IFG	75426	88.89

winner if it wins for even a single information state. To gain a better understanding of the large state problems where lp-best(ϵ) was the overall winner, we investigated what fraction of feasible pairs contain information points that favor lp-best(ϵ). Table 15.3 shows that for all problems except p5044, DC-NICHE wins for approximately 90% of feasible pairs. For p5044, DC-NICHE and lp-best(ϵ) roughly split the region.

Run time of the various heuristics is compared in Table 15.4. Since DC-NICHE and **IFG** all sample the information space, Table 15.5 compares the number of sample points necessary to obtain the results above.

7. Summary

The main contribution of this paper is the development of a **GA**-based procedure for determining high-quality, suboptimal designs for moderately sized finite horizon **POMDP**s. This **GA** is based on a random keys search of the information space stratified by a niching approach. This algorithm, DC-NICHE, was compared to two algorithms in the literature, **IFG** and lp-best, on 10 test problems.

Table 15.4. Comparisons among various algorithms: total run time

strategy	DC-NICHE	IFG	lp-best(ϵ)
examples	$\mathcal{T}_{0,20}^{GA}$	$\mathcal{T}_{0,20}^{IFG}$	$\mathcal{T}_{0,20}^{LP}$
p444	5.3	5.5	14.8
p533	1.7	1.7	6.6
p2044	2.5	4.3	14.3
p3044	14.8	35.2	73.9
p5044	175.5	173.5	3385.0
p10044	549.8	939.7	2159.1
p4204	57.1	53.8	199.1
p4304	241.2	302.6	742.2
p4504	298.9	335.9	142.7
p41004	643.2	673.3	229.8

Table 15.5. Comparisons among various algorithms: number of points evaluated

strategy	DC-NICHE	IFG	
example	$\Upsilon_{0,20}$	$\Upsilon_{0,20}$	d-values
p444	24314	30011	12-27
p533	14172	19052	8-13
p2044	9737	30800	3
p3044	12842	76725	2-3
p5044	24355	25500	2
p10044	25634	96050	1-2
p4204	17041	19239	11-17
p4304	24868	27835	11-20
p4505	20732	25219	11-19
p41004	21996	26323	11-19

Solution quality is difficult to measure for this problem since the output is not a single value, but a function mapping information state to value. Our first comparisons were very conservative in that they heavily favored lp-best. We measured the minimum advantage DC-NICHE had over lp-best for any information state. Hence, if lp-best won for a single information state, it would be declared the overall victor. Despite this conservative approach, DC-NICHE produced superior solutions on six of the ten problems. That is, DC-NICHE produced a better solution for every information state for each of these six problems. For the four problems where lp-best produced a superior solution for some information state, we did a detailed analysis of all information states. For these four problems, we found that DC-NICHE produced better solutions for, on average, 79% of the information states.

Comparing DC-NICHE to the fixed grid approach, IFG, was a closer comparison. In head-to-head comparison, DC-NICHE held a slight four wins, three losses, three ties advantage. The average fitness values produces favored DC-NICHE, but not to a significant level.

In computation speed, DC-NICHE had a much more substantial advantage. It solved the suite of ten problems in 1990.0 seconds, compared to 2525.5 seconds for **IFG** and 6967.5 for lp-best.

In conclusion, DC-NICHE produced, on average, better solutions than **IFG** or lp-best on these test problems, and did it in significantly less time.

Acknowledgments

This work was supported by the National Science Foundation under Grant DMI-9634712. We would like to thank Lidore Amit and Erin Eisenberg for running the computational tests and for other contributions to this paper. We would also like to thank Anthony Cassandra for his sharing the source codes for solving the **POMDP.**

References

Bean, J. C. (1994). Genetic algorithms and random keys for sequencing and optimization. *ORSA Journal of Computing,* 6 (2):154–160.

Cassandra, A. R., Littman, M. L., and Zhang, N. L. (1997). Incremental pruning: a simple, fast, exact algorithm for partially observable Markov decision processes. Technical report, Dept. of Computer Science, Brown University.

Cavicchio, Jr., D. J. (1970). *Adaptive Search Using Simulated Evolution.* PhD thesis, University of Michigan, Ann Arbor.

Charnes, A. and Cooper, W. W. (1962). Programming with linear fractional functionals. *Naval Research Logistics Quarterly,* 9:181–186.

Cheng, H.-T. (1988). *Algorithms for partially observable Markov decision processes.* PhD thesis, University of British Columbia, British Columbia, Canada.

Davidor, Y. (1991). A naturally occurring niche and species phenomenon: the model and first results. In *Proceedings of the Fourth International Conference on Genetic Algorithms.* Morgan Kaufmann.

Davis, L. (1991). *Handbook of Genetic Algorithms.* Van Nostrand.

De Jong, K. A. (1975). *An analysis of the behavior of a class of genetic adaptive systems.* PhD thesis, University of Michigan.

Eagle, J. N. (1984). The optimal search for a moving target when the search path is constrained. *Operations Research,* 32:1107–1115.

Fogel, L. J. (1962). Autonomous automata. *Industrial Research,* 4:14–19.

Fogel, L. J., Owens, A., and Walsh, M. J. (1965). Artificial intelligence through a simulation of evolution. In *Proceedings of the 2nd Cybernetic Sciences Symposium,* pages 131–155. Spartan Books, Washington.

Goldberg, D. E. (1989a). Genetic algorithms and Walsh functions: Part I, a gentle introduction. *Complex Systems,* 3:129–152.

Goldberg, D. E. (1989b). Genetic algorithms and Walsh functions: Part II, deception and its analysis. *Complex Systems,* 3:153–171.

Goldberg, D. E. (1989c). *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison-Wesley Publishing Company, Inc.

Goldberg, D. E. (1991). Construction of high-order deceptive functions using low-order Walsh coefficients. *Annals of Mathematics and Artificial Intelligence,* 5:35–48.

Goldberg, D. E. (1993). Making genetic algorithms fly: a lesson from the wright brothers. *Advanced Technology for Developers,* 2:1–8.

Goldberg, D. E., Deb, K., and Horn, J. (1992). Massive multimodality, deception, and genetic algorithms. In Männer, R. and Manderick, B., editors, *Parallel Problem Solving from Nature,* Amsterdam: North-Holland, 2, 37–46.

Goldberg, D. E. and Richardson, J. J. (1987). Genetic algorithms with sharing for multimodal function optimization. In *Proceedings of 2nd International Conference on Genetic Algorithms, Lawrence Erlbaum Publishers.*

Gorges-Schleuter, M. (1990). Explicit parallelism of genetic algorithms through population structure. Parallel Problem Solving from Nature, Springer-Verlag.

Hadj-Alouane, A. B., Bean, J. C., and Murty, K. G. (1999). A hybrid genetic/optimization algorithm for a task allocation problem. *Journal of Scheduling,* 2:189–201.

Holland, J. H. (1975). *Adaptation in Natural and Artificial Systems.* The Universiy of Michigan.

Holland, J. H. (1992). *Adaptation in Natural and Artificial Systems.* Cambridge, MA: MIT press.

Horn, J. and Goldberg, D. E. (1995). Genetic algorithms difficulty and the modality of fitness landscapes. In Whitley, L. D. and Vose, M. D., editors, *Foundations of Genetic Algorithms,* Morgan Kaufmann, San Mateo, CA, 3, 243–269.

Lark, J. W. (1989). *A heuristic search approach for solving finite horizon, completely unobserved Markov decision processes.* PhD thesis, University of Virginia.

Lin, A. Z.-Z. (1999). *A hybrid genetic/optimization algorithm for a class of sequential decision models.* PhD thesis, University of Michigan, Michigan, U.S.A.

Littman, M. L., Cassandra, A. R., and Kaelbling, L. P. (1995). Efficient dynamic programming updates in partially observable Markov

decision processes. Technical Report CS-95-19, Dept. of Computer Science, Brown University.

Lovejoy, W. S. (1991). Computationally feasible bounds for partially observed Markov decision processes. *Operations Research,* 39:162–175.

Mahfoud, S. W. (1995). *Niching method for genetic algorithms.* PhD thesis, University of Illinois at Urbana-Champaign.

Michalewicz, Z. (1996). *Genetic Algorithms + Data Structures = Evolution Programs.* Springer-Verlag.

Monahan, G. E. (1982). A survey of partially observable Markov decision processes: theory, models, and algorithms. *Management Science,* 28:1–16.

Murty, K. G. (1983). *Linear Programming.* John Wiley & Sons, Inc.

Norman, B. A. and Bean, J. C. (1997). A random keys genetic algorithm for job shop scheduling. *Engineering Design and Automation,* 3:145–156.

Norman, B. A. and Bean, J. C. (1999). A genetic algorithm methodology for complex scheduling problems. *Naval Research Logistics,* 46:199–211.

Norman, B. A. and Bean, J. C. (2000). Operation scheduling for parallel machine tools. *IIE Transactions,* 32:449–459.

Platzman, L. K. (1980). Optimal infinite-horizon undiscounted control of finite probabilistic systems. *SIAM Journal of Control Optimization,* 18:362–380.

Shaefer, C. G. and Smith, S. J. (1988). The argot strategy ii: Combinatorial optimizations. Technical report, Thinking Machine Corporation.

Smallwood, R. D. and Sondik, E. J. (1973). The optimal control of partially observable Markov decision processes over a finite horizon. *Operations Research,* 21:1071–1088.

Sondik, E. J. (1971). *The optimal control of partially observable Markov processes.* PhD thesis, Standford University, California, USA.

Sondik, E. J. (1978). The optimal control of partially observable Markov decision processes over the infinite horizon: discounted costs. *Operations Research,* 24:282–304.

Spears, W. M. and De Jong, K. A. (1991). On the virtues of parameterized uniform crossover. In *Proceedings of the Fourth International Conference on Genetic Algorithms,* 230–236.

Syswerda, G. (1989). Uniform crossover in genetic algorithms. In *Proceedings of the Third International Conference on Genetic Algorithms,* 2–9.

White, D. J. (1985). Real applications of Markov decision processes. *Interfaces,* 15:7–83.

White, D. J. (1988). Further real applications of Markov decision processes. *Interfaces,* 18:55–61.

White, III, C. C. (1991). A survey of solution techniques for the partially observed Markov decision processes. *Annals of Operations Research,* 32:215–230.

White, III, C. C. and Scherer, W. T. (1989). Solution procedures for partially observed Markov decision processes. *Operations Research,* 37:791-797.

White, III, C. C. and Scherer, W. T. (1994). Finite-memory suboptimal design for partially observed Markov decision processes. *Operations Research,* 42:439-455.

White, III, C. C. and White, D. J. (1989). Markov decision processes. *European Journal of Operations Research,* 39:1–16.

Whitley, D. (1991). Fundamental princlples of deception in genetic search. In Rawlins, G. J. E., editor, *Foundations of Genetic Algorithms,* pages 221–241. Morgan Kaufmann, San Mateo, CA.

Appendix

A. Let p_1 and p_2 be two chosen parents, and c_1 and c_2 be the two offspring created by the *composite crossover operator* applied to p_1 and p_2. Let $\alpha, \alpha_1, \alpha_2 \in [0, 1]$ be three randomly generated scalars. The α is termed as favorable tendency between parents. The composite crossover operator is defined as follows:

Crossover Step: For each gene $i \in \mathbf{S}$, do the following:

Step 1 Randomly generate three values: $\alpha, \alpha_1, \alpha_2$.

Step 2 Randomly perform either uniform crossover or linear combination crossover between $p_1(i)$ and $p_2(i)$ to create $c_1(i)$ and $c_2(i)$.

Uniform Crossover: Set $c_1(i) = \alpha_1 p_1(i)$ and $c_2(i) = \alpha_2 p_2(i)$ if $\alpha \leq 0.7$; otherwise, set $c_1(i) = \alpha_1 p_2(i)$ and $c_2(i) = \alpha_2 p_1(i)$.

Linear Combination: Set $c_1(i) = \alpha_1 p_1(i) + (1 - \alpha_1) p_2(i)$ and $c_2(i) = \alpha_2 p_1(i) + (1 - \alpha_2) p_2(i)$ if $\alpha \leq 0.7$; otherwise, set $c_1(i) = (1 - \alpha_1) p_1(i) + \alpha_1 p_2(i)$ and $c_2(i) = (1 - \alpha_2) p_1(i) + \alpha_2 p_2(i)$.

Perturbation Step Perform *give_N_share* operator on the two offspring. In *give_N_share* operator, each gene gives away some random portion of its alleles, and at the same time also receives some alleles from the other genes.

B. Data sets for test examples are listed in Table A.1. The examples p2044, p3044, p5044, and p10044 are referred to as **S**-examples as the only parameter among these examples to change is the cardinality of the state space **S.** Similarly, the examples p4204, p4304, p4504, and p41004 are referred to as **Z**-examples.

C. General parameters for the **GA** are listed in Table A.2, where $\mathcal{C}_v(\bullet)$ is defined in Table A.3. The maximum number of generations for stage t is defined as: $G_0 + (T - t) \times \triangle G$.

Table A.1. Data setting for 10 Test Examples

Example	#State	#Action	#Observation	Discount
p444 (Cheng, 1988)	4	4	4	1.00
p533 (Cheng, 1988)	5	3	3	1.00
p2044	20	4	4	0.87
p3044	30	4	4	0.90
p5044	50	4	4	0.95
p10044	100	4	4	0.875
p4204	4	4	20	0.87
p4304	4	4	30	0.90
p4504	4	4	50	0.95
p41004	4	4	100	0.875

Table A.2. Parameter settings for DC-NICHE.

Parameter	Value
distribution of points	uniform over hypercube
offspring-feasibility-ensuring scheme	random keys approach
fitness evaluation function	$\mathbf{F}(\mathbf{x}, \Gamma_t^{\Psi}, \Gamma_t^{\mathbf{GA}})$
fitness threshold function	$\mathcal{C}_v(\bullet)$
minimal population size	100
initial number of generations, G_0	10
number of extra generations, $\triangle G$	3
Mutations ratio	1%
Move_To_Boundary ratio	1%
partially-guided search ratio	1%
Favorable tendency Among Parents %	70
Return rate threshold, M_{r_rate}	0.05
Maximal consecutive zero runs, M_{0_run}	2

Table A.3. Definition for the fitness threshold function $\mathcal{C}_v$

$\lvert\Gamma\rvert$	$\mathcal{C}_v(\lvert\Gamma\rvert)$
$0-100$	$10^{-9}+(10^{-10}-10^{-11})\lvert\Gamma\rvert$
$100-150$	$10^{-8}+2(10^{-9}-10^{-10})(\lvert\Gamma\rvert-100)$
$150-200$	$10^{-7}+2(10^{-6}-10^{-9})(\lvert\Gamma\rvert-150)$
$200-300$	$10^{-4}+(10^{-5}-10^{-6})(\lvert\Gamma\rvert-200)$
$300-400$	$10^{-3}+4\times10^{-5}(\lvert\Gamma\rvert-300)$
$400-$	5×10^{-3}

Chapter 16

USING GENETIC ALGORITHMS TO FIND GOOD *K*-TREE SUBGRAPHS

Elham Ghashghai and
Ronald L. Rardin

Abstract Many combinatorial problems which are (*NP*) hard on general graphs yield to polynomial algorithms when restricted to k-trees which are graphs that can be reduced to the k-complete graph by repeatedly removing degree k vertices having completely connected neighbors. We present a genetic algorithm which seeks a heuristic optimum solution by generating an evolving population of k-tree subgraphs. Each is evaluated by computing an exact optimum over the subgraph, which provides a feasible solution over the original graph. Then we validate our algorithm by testing it on the task of finding a minimum total cost 3-tree in a complete graph.

1. Introduction

Genetic Algorithms (GAs) form a class of heuristic searches that imitate natural selection. They maintain an evolving collection or population of solutions throughout the search. The basic concepts of GAs were developed by Holland in 1975 (Holland, 1975). Goldberg (1989) comments that GAs are more efficient and flexible than other traditional heuristic search methods since GAs start from a set of solutions and not a single one, and also because they use the objective function value as opposed to other auxiliary knowledge (Goldberg, 1989).

In the last few years there has been an increasing interest in using GAs for solving a wide variety of optimization problems due to their efficiency, robustness and flexibility. On the other hand, the richness and decomposibilty of k-trees have been very attractive to many researchers for solving a variety of combinatorial optimization problems that are known to be (*NP*) hard on general graphs, but can be solved in poly-

nomial time when restricted to k-trees (Arnborg et al., 1997; Arnborg and Proskurowski, 1986; Arnborg and Proskurowski, 1989; Coullard et al., 1991; Coullard et al., 1991b; Granot and Skorin-Kapov, 1988; Wald and Colbourn, 1982).

The problem with this approach is that a given graph may not be a k-tree (for example a complete graph), and even when a graph is known to be a k-tree, determining the size of k (width of the graph) is itself (*NP*) hard (Bienstock, 1991).

Our research aims at using genetic algorithms to find a "good" k-tree subgraph in a general graph. The optimal solution on this subgraph is an approximation of the optimal solution on the complete graph. In particular, we focus on *Strong k-Connectivity Problem* (SKCP) which seeks a minimum cost subgraph of a general graph in which any sites or links can fail as long as no k of them are mutually adjacent. This problem is equivalent to finding a minimum cost k-tree, which is a generalization of IFI networks. An IFI network is one in which any sites or links can fail as long as no two of them are adjacent. Farely, Wald and Colbourn show that this problem is in fact a 2-tree (Wald and Colbourn, 1982). Beltran et al. heuristicly construct minimum cost IFI (2-tree) networks (Beltran and Skorin-Kapov, 1994). In general a k-tree is a network such that any sites or links can fail as long as no k of them are adjacent.

In this chapter we focus on the application of our algorithm to 3-trees. The extension to k-trees is straightforward. In Section 2 definitions of k-trees and decomposition trees are given. In Sections 3 and 4, we discuss the algorithm paradigm and implementation in detail. Computational results are discussed in Section 5.

2. k-Trees

Graph theoretic terminology used here generally follows Bondy and Murty (1976). A special class of graphs known as *k-trees* is defined recursively as follows. A k-tree on k vertices is a k-clique (complete graph on k vertices). If G is a k-tree on n vertices, then we can construct G', a k-tree on $n+1$ vertices, by making the $(n+1)$st vertex adjacent to each vertex of a k-clique of G. Equivalently we can say that, k-trees are *perfect elimination* graphs, that is, graphs $G = (V, E)$ for which there exists an ordering of the $V(G)$ such that each vertex forms a $k+1$ clique with its neighbors just before it is eliminated (Rose, 1974).

Figure 16.1 illustrates the construction of a 3-tree. A 3-clique (i.e. a triangle) is the most elementary 3-tree. To construct a 3-tree on four vertices, connect a fourth vertex to each vertex of the original triangle. This will create a simplex with four triangles. To add a fifth, connect

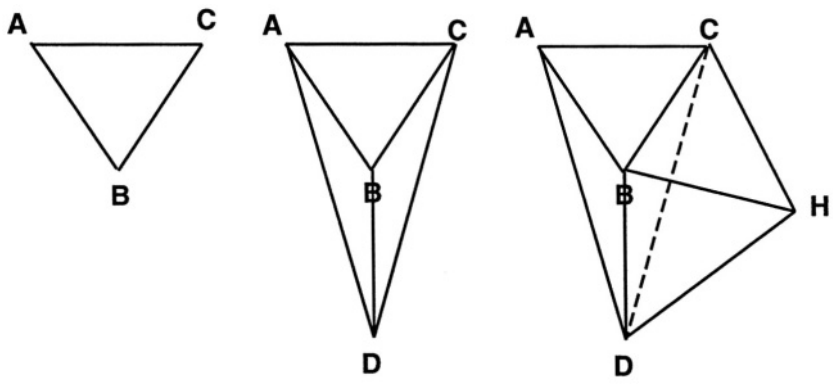


Figure 16.1. 3-tree Construction

another vertex to each vertex of any one of the existing triangles, an so forth.

Partial k trees are edge subgraphs of *k*-trees. In developing linear-time algorithms for 3-trees it is sufficient to address only full (maximal) cases, because a partial 3-tree can be embedded in a (complete) 3-tree in linear time (Matoušek and Thomas, 1991).

2.1 Decomposition Trees

A graph $G(V, E)$ is a *k*-tree if and only if every minimal separator of any distinct non-adjacent vertices is a *k*-clique (Rose, 1974). For example, the triangle *BCD* separates vertices *A* and *H* in the 3-tree of Figure 1.

This property leads naturally to a decomposition of any 3-tree into simplices that intersect in separating triangles. Figure 16.2 illustrates a *decomposition tree* of a 3-tree on eight vertices. Such a decomposition can be constructed in linear time (Corneil and Keil, 1987). An enumeration of the nodes in any decomposition tree can be obtained by a breadth search. Later in this paper we use this characteristic of *k*-trees to encode them for genetic search.

3. Algorithm Paradigm and Terminology

Genetic Algorithms operate on a set (usually of fixed size) of solutions, called a *population* which evolves through a sequence of *generations.* The first set of solutions which is randomly generated is referred to as the *initial population.* The process moves from one generation to another by breeding new solutions. Transitions from one generation to the next are

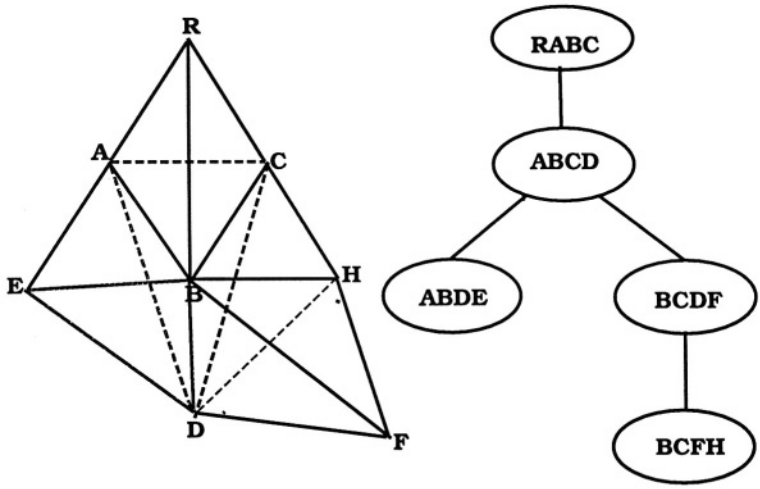


Figure 16.2. Decomposition Tree

based on the objective function values, i.e., the best solutions stay, and more are generated by breeding.

There are many variations of genetic algorithms. They differ mainly in their implementations of three essential operators; reproduction, crossover and mutation. The details must therefore be expressed for each algorithm, and we do so in the next section.

3.1 Transition From One Generation to the Next

Reproduction occurs through copying a fixed number of best solutions (elite) from the current generation into the next. This is called an *elitist strategy.*

Crossover is a process by which two solutions combine to create two new offspring. In this paper we use a *one-point* crossover scheme as follows. Assume that each solution is somehow encoded as a string of D entries. We pick an integer p, the *crossover point,* randomly from $(1,2, \ldots, D)$. In a one-point crossover, the two parent strings break at the crossover point, and switch tails to create two offspring. The parents are annihilated in the process. For example consider two sequences of nodes, $(1,2,3,4,5,6,7)$ and $(7,6,5,3,1,2,4)$. Assume that the crossover point is $p = 2$. Figure 16.3 shows the two offspring after crossover: $(1,2,5,3,1,2,4)$ and $(7,6,3,4,5,6,7)$.

The above crossover operator may produce invalid strings. For example, if strings are supposed to be a permutation of a set of nodes, then

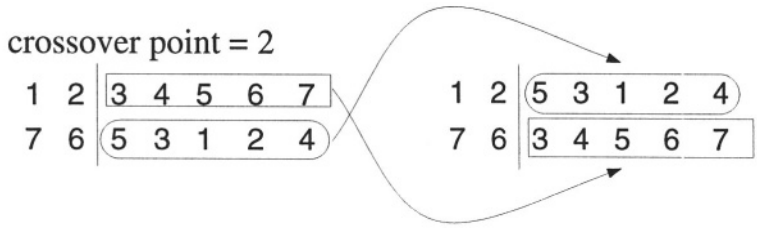


Figure 16.3. Crossover with Infeasible Offspring

a string containing repeated nodes is invalid, as happens in the previous example.

To overcome this problem we represent a D-sequence with a sequence of D random numbers from [0,1]. Such a sequence is known as a *random key* (Bean, 1994). The mapping from a random key to the original sequence is accomplished by first enumerating the components of the key in ascending order. The original sequence will be the respective positions of the random numbers in the enumeration. For example, consider the key (.80, .31, .62, .01, .42). In this sequence .01 is the smallest number and is in position 4. Therefore, node 4 is the first component of the node sequence. Altogether, (.80, .31, .62, .01, .42) represents the sequence (4,2,5,3,1). We perform the one-point crossover on the random keys. The offspring will then be mapped back to node sequences.

The elitist strategy is prone to premature convergence where all or nearly all the population consists of identical solutions. Traditionally, this problem is overcome by a *mutation* process which occasionally and randomly alters the value at a string position.

In our research we used, instead, Bean's approach of introducing a few randomly generated solutions to each generation(Bean, 1994). This operation, known as *immigration,* slows down the process of population convergence by adding diversity. Immigration was chosen because it avoids infeasible encodings which can easily arise from random changes as we will see in Section 4. The same routines used to generate the initial population produce random immigrants. Immigration also provides a basis for evaluating solutions as we will see in Section 5.

Figure 16.4 schematically shows the transition from one generation to the next. The evolutionary process is eventually truncated when improvement falls below a specified level.

4. Genetic Algorithm Implementation

A critical issue in applying a genetic algorithm is to devise an encoding of the solutions that is capable of spanning the solution set without

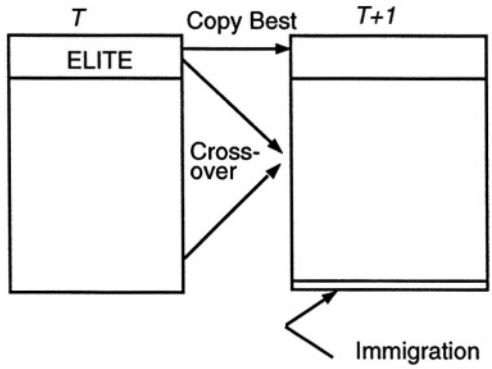


Figure 16.4. Generation T to Generation $T+1$

including infeasible ones. Also the encoding should be unbiased, i.e., all solutions should be equally likely. Another important consideration is to avoid infeasibility of offspring after crossover to avoid the use of penalty costs. Still other issues are implementation parameters such as population size, the ratio of elite, immigration and crossover, and the stopping criteria which affect the efficiency of the implementation.

4.1 Encoding k-tree Subgraphs

Consider the decomposition tree discussed in Section 2.1. A natural way to encode a k-tree and in particular a 3-tree is to present the topology of a decomposition and its sequence of nodes (see Figure 2). These representations are called the *topology* and *sequence strings* respectively. We represent the topology of any decomposition tree by a string of k-tuples with binary components. The jth k-tuple in the string corresponds to the jth node in the decomposition tree taking nodes in breadth first sequence. Each component in a k-tuple represents a child (or lack thereof) in the respective position for the corresponding node. That is, 1's show where new vertices are inserted to creat children as the 3-tree grows. For example, assume that the triple (110) is in the 3rd position of the string. It means the 3rd node in the decomposition tree has a first and a second child and no third. We use the notations S_N and T_D for the sequence and topology strings respectively, where N is the number of vertices in the graph and D is the number of nodes (k-cliques) in the decomposition graph.

As an example, consider Figure 16.2 again. Start with a simplicial node as the root, R. $ABRC$ is the root simplex. The specific choice of the sequence of letters will be justified later. Now R being the root, the

next vertex, *D,* will have to be connected to the *ABC* face, and *ABCD* has to be the first (and only) child of the root simplex. This corresponds to topology string (100) with the sequence string (*ABRCD*). The simplex *ABCD* has three available free faces. By that we mean that we can add a new node to any of the faces except the *ABC* face. It is a matter of convention how we number the children.

In our algorithm we make a convention that the next child will be a first child, if it is connected to the *ABD* face, a second child if connected to the *ACD* face, and a third, if connected to the *BCD* face. This also justifies the naming of the root simplex, in as much as it can have only one child. In Figure 2, the next vertices, *E* and F, are connected to *ABD* and *BCD* respectively. There is no third child for this simplex, hence our sequence and topology strings are (*ABRCDEF*) and (100, 110). The simplex *ABDE* has no child, *BCDF* has only a first child *BCFH,* and *BCFH* has no child. Thus the complete topology and sequence strings for this graph are T_5 = (100, 110,000, 100,000) and S_8 = (*ABRCDEFH*) respectively.

4.2 Generating Decomposition Trees Randomly

As mentioned in the previous section, the initial population should be a "good" representative of the whole population space. To achieve this, we should be able to generate decomposition trees with all types of topologies. In other words, the likelihood of getting a narrow tree should be comparable to that of getting a bushy one. Our prototype is a decomposition tree for a 3-tree, but our algorithm can be easily generalized to k-trees by simply changing 3 to k for any fixed k.

We generate a topology string in two stages. First we generate a sequence (C_n) Where $C_n \in \{0,1,2,3\}$ is the number of children of node n in the decomposition tree. This is done iteratively, starting from the root node. In each iteration, we have to decide to which of the existing nodes (potential parents) we must assign the next node. The sequence is complete when all the nodes (except the root) have been assigned to some parent. In the second stage, for each C_n, we randomly generate a triple of C_n 1's and the rest 0's. This will determine the position of each child as described earlier.

A step by step description of the first stage is as follows. At the outset of the problem, we generate 3 unif[0,1] numbers W_0, W_1 and W_2. These numbers will determine the proportion of nodes with 0, 1, 2 or 3 children. At each iteration, we assign the weight W_i to any node that has already been assigned i children. Each node will then have a chance of receiving the next child proportional to the weight assigned to it. For

notational convenience, we let $W_3 = 0$, signifying the fact that a node with 3 children cannot have another child.

More precisely, suppose that by iteration t we have established the parent-child relation of the first t nodes. If node $n \in \{1, \ldots, t\}$ has been assigned $n_j \in \{0, 1, 2, 3\}$ children, then we assign to it the weight W_{n_j}. In iteration $(t+1)$, node $(t+1)$ will become a child of node n with probability $p^{(n)} = W_{n_j} / \sum_{k=1}^{t} W_{k_j}$. Notice that in the competition for the next child, the weights W_i determine which type of parents have a better chance.

4.3 Crossover Operator

The intuition behind the crossover operator is that it mixes characteristics of the parents. Such characteristics could be as specific as a particular arrangement of the edges or as vague as the general shape of the parent trees. Consider the example depicted in Figure 16.5. The decomposition trees, ordered in depth first search, are cut along with their node sequences from the crossover point and the tails are swapped. On the one hand, offspring 1 and 2 inherit intact sections of parents 1 and 2. On the other hand, the first offspring is narrower than parent 1 while the second one is bushier than parent 2. One hopes that good characteristics will survive and evolve through generations because the best (elite) solutions have more chance to reproduce.

The concept is translated to the following procedure. The topology and sequence strings are crossedover simultaneously. We crossover the two topology sequences after p 1s (simplices), which corresponds to $p+3$ vertices in the sequence string. We also crossover the random keys for the parents, and decode the new keys to get the new sequence strings. Figure 16.6 illustrate an example when $p = 2$.

5. Computational Results

Our computational results are based on applying the algorithm to the problem of a minimum total cost 3-tree on three sets of complete graphs of sizes 30, 50 and 80 nodes. Nodes were generated randomly as independent and identically distributed points (x, y) uniformly over $[0, 1000] \times [0,1000]$. The underlying networks are complete graphs in the Euclidean plane, and arc costs are the lengths. Programming was done in C and all computation times are reported in seconds on Sun/Sparc.

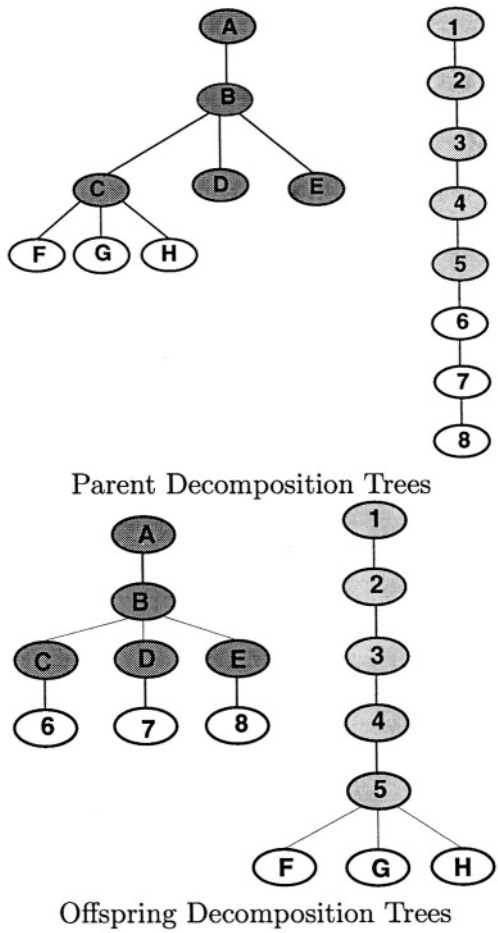


Figure 16.5. Decomposition Tree-Based Crossover Scheme

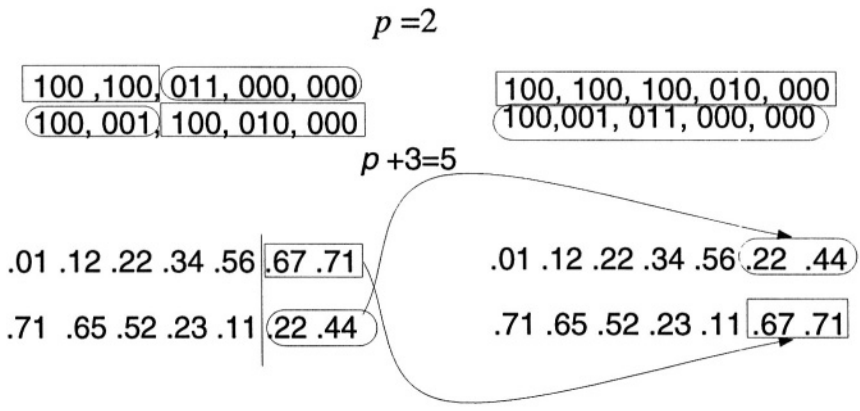


Figure 16.6. Crossover Topology and Sequence Strings

5.1 Preliminary Experiment

Preliminary experimentation suggested that the algorithm is not sensitive to moderate changes in the elite and immigration ratios as long as the fractions are about .20 and .03 respectively. These numbers are also consistent with the literature (Bean, 1994) of elitist strategies.

To find a suitable population size, we tested the algorithm with different population sizes on three problem instances for each problem size, each with five different runs starting from a different initial population. Table 5.1 gives a summary of performance for each case.

We chose our population sizes as scalar multiples of $\sqrt{N}$, where N is the number of nodes. We tested multiplications 5, 10, 20 and 25. For each problem size, the number of function evaluations was fixed. Thus bigger populations implied fewer generations.

There were a total of $4 \times 5 = 20$ runs for each problem instance. The minimum over all 20 runs is the best known solution used for comparison purposes. For each problem instance, the average, minimum and maximum percent errors from the best known are also reported in Table 5.1. The errors decreased as the population size increased from $5\sqrt{N}$ to $20\sqrt{N}$, and increased again for higher population sizes. Therefore, we fixed a population size of $20\sqrt{N}$ for all further investigations.

5.2 Main Experiment

Our main experiment consisted of applying our algorithm to the problem of a minimum total cost 3-tree on three sets of complete graphs of sizes 30, 50 and 80 nodes. Five problem instances of each of the above sizes were randomly generated, and each problem was run with 10 dif-

Table 16.1. Impact of Population Size on Solution Quality

		Error vs. Best (%)			
Pop. ($\times\sqrt{N}$)		5	10	15	20
Nodes = 30					
Ins 1	Avg	9.02	6.17	4.98	6.19
	Min	4.18	1.84	2.20	0.00
	Max	17.61	15.15	8.64	15.13
Ins 2	Avg	12.99	11.53	7.84	14.26
	Min	4.57	5.20	0.00	4.99
	Max	20.91	16.17	13.51	22.99
Ins 3	Avg	10.91	9.13	8.50	13.26
	Min	5.17	6.40	0.00	4.99
	Max	16.56	10.15	15.36	20.99
Overall Avg		10.97	8.94	7.11	11.24
Nodes = 50					
Ins 1	Avg	8.55	8.11	5.18	3.78
	Min	2.51	2.96	0.00	0.41
	Max	13.74	14.22	9.19	6.70
Ins 2	Avg	8.11	5.18	3.78	4.69
	Min	2.96	0.00	0.41	0.00
	Max	14.22	9.19	6.70	10.17
Ins 3	Avg	9.83	7.16	8.59	10.34
	Min	3.47	2.48	2.68	5.16
	Max	12.01	10.95	13.41	17.20
Overall Avg		8.83	6.82	5.85	6.27
Nodes = 80					
Ins 1	Avg	5.75	6.49	4.25	6.74
	Min	0.00	5.20	0.40	4.00
	Max	13.19	8.04	6.58	10.11
Ins 2	Avg	13.69	10.28	8.01	3.80
	Min	12.02	5.03	2.06	0.00
	Max	17.80	15.41	12.20	9.93
Ins 3	Avg	2.83	4.90	5.53	6.90
	Min	0.44	0.15	0.20	0.01
	Max	4.66	7. 85	9.66	12.03
Overall Avg		7.42	7.22	5.93	5.81

Table 16.2. Number of Generations in Main Experiment

Prob Size		30	50	80
Ins 1	Avg	122	471	578
	Min	56	345	314
	Max	227	645	1123
Ins 2	Avg	177	406	600
	Min	92	166	140
	Max	256	696	956
Ins 3	Avg	166	391	612
	Min	104	166	189
	Max	261	707	1133
Ins 4	Avg	189	349	761
	Min	116	98	304
	Max	255	631	1137
Ins 5	Avg	154	437	657
	Min	51	219	282
	Max	261	656	1131
Overall Avg		161	411	641

ferent initial populations. Each run was truncated after objective value stabilized and no improvement occurred. Table 5.2 reports the average, minimum and maximum number of generations before the algorithm converged. The CPU times for generating and processing each generation in 30, 50 and 80 nodes were .18, .26, 1.40 seconds respectively.

5.3 Heuristic Solution Quality

One of the most difficult issues in empirical testing of heuristics is to estimate how close heuristic solutions come to true optimal values. We evaluated our algorithm in three ways.

First, we compared our results to a purely random search. For each problem instance in 30, 50 and 80 nodes, 90,000, 300,000 and 2,600,000 random feasible solutions were generated. Random solutions for each problem were divided into ten batches and the best objective value of each batch was found. Figure 16.7 shows the distribution of the percent error of the best random solutions against our heuristic. GA's best found solution is superior to the best random solutions by at least 25%.

We also used Golden and Alt's (1979) procedure to estimate the global minimum. Given n independent samples from a specific instance of

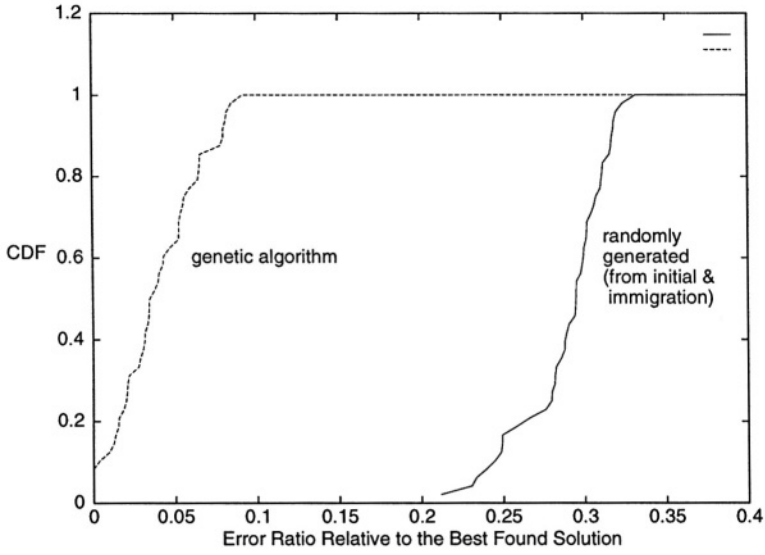


Figure 16.7. Cumulative Distribution of Errors in Main Experiment

a (minimizing) combinatorial optimization problem, with each sample consisting of m solution values, we can obtain an estimate and the corresponding confidence interval on the true optimal value z^*. Specifically, suppose we order the sample minima $z_{[1]} \leq z_{[2]} \leq \ldots \leq z_{[n]}$. Then for large m and n the distribution of the least sample minimum $z_{[1]}$ is approximately Weibull. Derigs (1985) derives the confidence interval with $\text{Prob}\{z_l \leq z^* \leq z_u\} \approx 1 - e^{-n}$ where $z_l = (z_{[1]} - \hat{b})$ and $z_u = z_{[1]}$. The scale and location parameters a and b are estimated by

$$\hat{a} = \frac{z_{[1]}z_{[n]} - (z_{[2]})^2}{z_{[1]} + z_{[n]} - 2z_{[2]}}$$

$$\hat{b} = z_{[\lfloor .63n+1 \rfloor]} - \hat{a}$$

and the shape parameter c is irrelevant.

Starting with 10 random initial populations and applying the genetic algorithm, we obtained $n = 10$ best solutions. Each is the minimum of all solutions produced during the run, whether retained in the population or not, and the values are presumably independent sample points because of the random start. Using the above method to estimate the global minimum, the average percent error of our best solutions to the estimated true minimum was 6.11%, 4.21% and 4.08% for problem sets of size 30, 50 and 80 respectively.

Table 16.3. Percent Error Versus The Best Found Solution in Main Experiment

Prob Size		30	50	80
Ins 1	Avg	5.17	3.25	2.76
	Max	11.80	5.97	5.26
Ins 2	Avg	5.00	5.60	3.56
	Max	12.51	13.27	7.98
Ins 3	Avg	8.50	6.88	6.28
	Max	15.88	8.90	10.19
Ins 4	Avg	4.75	2.33	4.12
	Max	8.80	5.16	9.51
Ins 5	Avg	5.94	3.99	4.12
	Max	12.90	7.60	6.20
Overall Avg		5.87	4.41	4.17

Finally, we compared each run with the best found solution over all 10 runs to measure robustness. In Table 16.7, the average and maximum errors relative to the best solution are given. The variability was 5.1% on the average. The overall conclusion is that our procedure results in error between 2.5 and 10% of the true optimum, which compared to the 25% error in randomly generated solutions is a considerable improvement.

6. Concluding Remarks and Further Research

The goal of this research was to develop a genetic-algorithm-based scheme for finding good k-trees. Results above suggest that the proposed methods are effective when a minimum total weight 3-tree is sought. Future research will seek to extend our approach to cases where a heuristic optimum for a more complicated combinatorial optimization is obtained by applying an optimal algorithm on each k-tree visited in the GA search and retaining the best solution found.

References

Arnborg, S., Corneil, D. G. and Proskurowski, A. (1997). Complexity of finding embeddings in a k-tree. *SIAM J. Alg. Disc. Math.,* 8, 277-284.

Arnborg, S. and Proskurowski, A. (1986). Characterization and recognition of partial 3-trees," *SIAM J. Alg. Disc. Math.,* 7, 305-314.

Arnborg, S. and Proskurowski, A. (1989). Linear time algorithms for *NP*-hard problems restricted to partial k-trees. *Disc. Appl. Math.,* 23, 11-24.

Bean, J. (1994). Genetic algorithms and random keys for sequencing and optimization. *ORSA journal on Computing,* 6, 154-160.

Beltran, H.F., and Skorin-Kapov, D. (1994). On minimum cost isolated failure immune networks. *Telecommunication Systems* 3, 2.

Bienstock, D. (1991). Graph searching, path-width, tree-width and related problems (a survey). *DIMA CS Ser. in Discrete Mathematics and Theoretical Computer Science,* 5, 33-49.

Bondy, J. A. and Murty, U.S.R. (1976). *Graph Theory with Applications,* North Holland.

Corneil, D. G. and Keil, J. M. (1987). A dynamic programming approach to the dominating set problem on k-trees. *SIAM J. Alg. Disc. Math.,* 8, 535-543.

Coullard, C. R., Rais, A., Rardin, R. L. and Wagner, D. K. (1991). The 2-connected Steiner subgraph polytope for series-parallel graphs. Technical Report 91-32, Purdue University, West Lafayette, Indiana.

Coullard, C. R., Rais, A., Rardin, R. L. and Wagner, D. K. (1991b). The dominant of the 2-connected Steiner subgraph polytope for W_4-free graphs. Technical Report 91-34, Purdue University, West Lafayette, Indiana.

Derigs, U. (1985). Using Confidence Limits for the Global Optimum in Combinatorial Optimization. *Operations Research,* 33, 1024-1049.

El-Mallah, E. S. and Colbourn, C. J. (1988). Partial k-tree algorithms. *Congressus Numerantium,* 64, 105-119.

El-Mallah, E. S. and Colbourn, C. J. (1990). On two dual classes of planar graphs. *Discrete Mathematics,* 80, 21-40.

Glover, F. and Laguna, M. (1993). in C. Reeves editor. *Modern Heuristic Techniques for Combinatorial Problems,* John Wiley.

Granot, D. and Skorin-Kapov, D. (1988). On some optimization problems on k-trees and partial k-trees. *Discrete Appl. Math.*.

Golden, B.L. and Alt, F.B. (1979). Interval Estimation of a Global Optimum for Large Combinatorial Problems. *Naval Research Logistics Quarterly, 26,* 69-77.

Goldberg, D. (1989). *Genetic Algorithms in Search, Optimization and Machine Learning,*. Addison-Wesley.

Holland, J.H. (1975). Adaptation in natural and artificial systems. *Ann Arbor: The University of Michigan Press.*

Matoušek, J. and Thomas, R. (1991). Algorithms finding tree-decompositions of graphs. *J. Algorithms,* 12, 1–22.

Rose, D. J. (1974). On Simple Characterizations of k-trees. *Discrete Mathematics,* 7, 317-322.

Wald, J. A. and Colbourn, C. J. (1982). Steiner Trees, Partial 2-trees, and Minimum IFI Networks. *Networks,* 13, 159-167.

This page intentionally left blank

Index