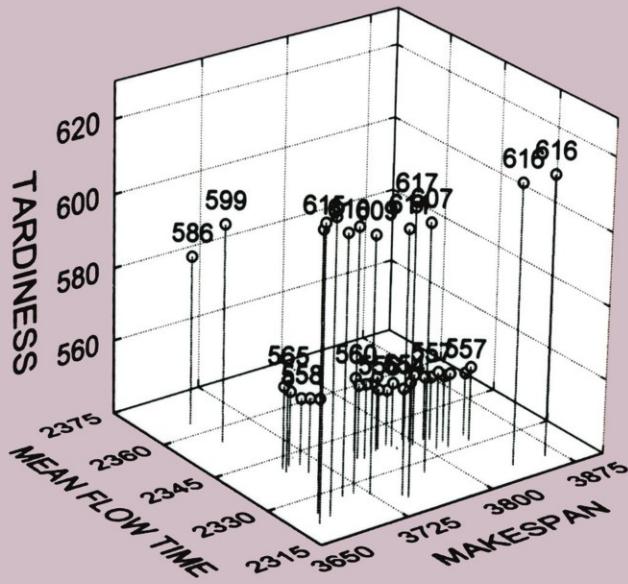


MULTIOBJECTIVE SCHEDULING BY GENETIC ALGORITHMS



Tapan P. Bagchi

Springer Science+Business Media, LLC

MULTIOBJECTIVE SCHEDULING BY GENETIC ALGORITHMS

MULTIOBJECTIVE SCHEDULING BY GENETIC ALGORITHMS

by

Tapan P. Bagchi

Indian Institute of Technology Kanpur



Springer Science+Business Media, LLC

Library of Congress Cataloging-in-Publication Data

Bagchi, Tapan P.

Multiobjective scheduling by genetic algorithms / by Tapan P. Bagchi.

p. cm.

Includes bibliographical references and index.

ISBN 978-1-4613-7387-2 ISBN 978-1-4615-5237-6 (eBook)

DOI 10.1007/978-1-4615-5237-6

1. Production scheduling --Computer simulation. 2. Genetic algorithms. 3. Multiple criteria decision making. I. Title.

TS157.5.B33 1999

658.5--dc21

99-37211

CIP

Copyright © 1999 by Springer Science+Business Media New York

Originally published by Kluwer Academic Publishers in 1999

Softcover reprint of the hardcover 1st edition 1999

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system or transmitted in any form or by any means, mechanical, photo-copying, recording, or otherwise, without the prior written permission of the publisher, Springer Science+Business Media, LLC.

Printed on acid-free paper.

This book is dedicated to

Chhoto Ma

And to the fond memory of

Chhoto Kaku

Table of Contents

	Page
Preface	xi
Chapter 1	Shop Scheduling: An Overview
1.1	What Is Scheduling?
1.2	Machine Scheduling Preliminaries
1.3	Intelligent Solutions to Complex Problems
1.4	Scheduling Techniques: Analytical, Heuristic and Metaheuristic
1.5	Outline of this Text
Chapter 2	What are Genetic Algorithms?
2.1	Evolutionary Computation and Biology
2.2	Working Principles
2.3	The Genetic Search Process
2.4	The Simple Genetic Algorithm (SGA)
2.5	An Application of GA in Numerical Optimization
2.6	Genetic Algorithms vs. Traditional Optimization
2.7	Theoretical Foundation of GAs
2.8	Schema Processing: An Illustration
2.9	Advanced Models of Genetic Algorithms
Chapter 3	Calibration of GA Parameters
3.1	GA Parameters and the Control of Search
3.2	The Role of the "Elite" who Parent the Next Generation
3.3	The Factorial Parametric Study
3.4	Experimental Results and Their Interpretation
3.5	Chapter Summary
Chapter 4	Flowshop Scheduling
4.1	The Flowshop
4.2	Flowshop Model Formulation
4.3	The Two-Machine Flowshop
4.4	Sequencing the General m -Machine Flowshop
4.5	Heuristic Methods for Flowshop Scheduling

4.6	Darwinian and Lamarckian Genetic Algorithms	92
4.7	Flowshop Sequencing by GA: An Illustration	97
4.8	Darwinian and Lamarckian Theories of Natural Evolution	99
4.9	Some Inspiring Results of using Lamarckism	102
4.10	A Multiobjective GA for Flowshop Scheduling	106
4.11	Chapter Summary	107
Chapter 5	Job Shop Scheduling	109
5.1	The Classical Job Shop Problem (JSP)	109
5.2	Heuristic Methods for Scheduling the Job Shop	117
5.3	Genetic Algorithms for Job Shop Scheduling	122
5.4	Chapter Summary	135
Chapter 6	Multiobjective Optimization	136
6.1	Multiple Criteria Decision Making	136
6.2	A Sufficient Condition: Conflicting Criteria	138
6.3	Classification of Multiobjective Problems	138
6.4	Solution Methods	139
6.5	Multiple Criteria Optimization Redefined	142
6.6	The Concept of Pareto Optimality and "Efficient" Solutions	143
Chapter 7	Niche Formation and Speciation: Foundations of Multiobjective GAs	147
7.1	Biological Moorings of Natural Evolution	148
7.2	Evolution is also Cultural	154
7.3	The Natural World of a Thousand Species	158
7.4	Key Factors Affecting the Formation of Species	160
7.5	What is a Niche?	161
7.6	Population Diversification through Niche Compacting	163
7.7	Speciation: The Formation of New Species	166
Chapter 8	The Nondominated Sorting Genetic Algorithm: NSGA	171
8.1	Genetic Drift: A Characteristic Feature of SGA	171
8.2	The Vector Evaluated Genetic Algorithm (VEGA)	173
8.3	Niche, Species, Sharing and Function Optimization	174
8.4	Multiobjective Optimization Genetic Algorithm (MOGA)	179

8.5	Pareto Domination Tournaments	179
8.6	A Multiobjective GA Based on the Weighted Sum	180
8.7	The Nondominated Sorting Genetic Algorithm (NSGA)	181
8.8	Applying NSGA: A Numerical Example	187
8.9	Chapter Summary	202
Chapter 9	Multiobjective Flowshop Scheduling	203
9.1	Traditional Methods to Sequence Jobs in the Multiobjective Flowshop	203
9.2	Disadvantages of Classical Methods	206
9.3	Adaptive Random Search Optimization	207
9.4	Recollection of the Concept of Pareto Optimality	207
9.5	NSGA Solutions to the Multiobjective Flowshop	209
9.6	How NSGA Produced Pareto Optimal Sequences	211
9.7	The Quality of the Final Solutions	214
9.8	Chapter Summary	214
Chapter 10	A New Genetic Algorithm for Sequencing the Multiobjective Flowshop	216
10.1	The Elitist Nondominated Sorting Genetic Algorithm (ENGA)	217
10.2	Initialization of ENGA (Box 1)	219
10.3	Performance Evaluation	220
10.4	Genetic Processing Operators	224
10.5	The <i>Additional</i> Nondominated Sorting and Ranking (Box 8)	230
10.6	Stopping Condition and Output Module	232
10.7	Parameterization of ENGA by Design of Experiments	232
10.8	Application of ENGA to the 49-Job 15-Machine Flowshop	237
10.9	Chapter Summary	237
Chapter 11	A Comparison of Multiobjective Flowshop Sequencing by NSGA and ENGA	245
11.1	NSGA vs. ENGA: Computational Experience	245
11.2	Statistical Evaluation of GA Results	247
11.3	Chapter Summary	249

Chapter 12 Multiobjective Job Shop Scheduling	256
12.1 Multiobjective JSP Implementation	256
12.2 NSGA vs. ENGA: Computational Experience	261
12.3 Chapter Summary	261
Chapter 13 Multiobjective Open Shop Scheduling	267
13.1 An Overview of the Open Shop	267
13.2 Multiobjective GA Implementation	268
13.3 NSGA vs. ENGA: Some Computational Results	270
Chapter 14 Epilog and Directions for Further Work	277
• Exact solutions	278
• Solving the General Job Shop	278
• Seeking Pareto Optimality	279
• Optimization of GA Parameters	279
• ENGA vs. Other Multiobjective Solution Methods	281
• Conflicting and Synergistic Optimization Objectives	283
• Darwinian and Lamarckian GAs: The High Value of Hybridizing	288
• Concluding Remarks	289
References	293
Appendix	C++ Codes for a Hybridized GA to Sequence the Single-Objective Flowshop
	307
Glossary	341
Index	351

Preface

This book describes methods for developing multiobjective solutions to common production scheduling situations modeled in the literature as flowshops, job shops and open shops. The methodology is metaheuristic, one inspired by how nature has evolved a multitude of coexisting species of living beings on earth.

Multiobjective scheduling situations are ubiquitous. Rarely is a shop manager interested only in "getting the orders out the fastest way possible." Typically, he/she attempts also to minimize tardiness, maximize the utilization of expensive capital equipment and human resources, minimize the mean flow time of the jobs to be done, etc. etc. In this book we demonstrate a framework for representing such challenging problems and then finding their solutions efficiently. The method uses enhancements of "genetic algorithms" (GAs)—search methods that use the "survival of the fittest" rule and cross-breeding, mutation and niche-formation, processes that nature is believed to have used to create well-adapted and co-habiting life forms.

In precise terms, we use enhancements of the *Nondominated Sorting Genetic Algorithm* (NSGA), a metaheuristic method recently proposed, which produces Pareto-optimal solutions to numerical multiobjective problems. One such important enhancement introduced in this text is called the *Elitist Nondominated Sorting Genetic Algorithm* (ENGA). The object of such methods is singular: solve a variety of multiobjective optimization problems, *and* do it efficiently. The final solutions evolved are all Pareto-optimal or "efficient." In this regard, these methods may be easily extended to other multiobjective decision situations such as configuring an FMS, operating an airport, or providing a multiplicity in patient care. Thus this book is intended for students of industrial engineering, operations research, operations management and computer science, as well as practitioners. It may also assist in the development of efficient shop management software tools for schedulers and production planners who face multiple planning and operating objectives as a matter of course.

The text begins with an overview of shop scheduling. A beginner's introduction to GAs and their parameterization is presented in Chapters 2 and 3. Flowshop scheduling is outlined in Chapter 4, including the key heuristic rules to solve the single-objective flowshop. We then discuss the use of GAs to sequence a flowshop, including methods for hybridizing GAs to make them efficient.

Job shop scheduling, a problem in which the floor routing of the different jobs varies, is reviewed in Chapter 5. Important solution methods are summarized, including mathematical programming formulations and the noteworthy heuristic rules. Recent methods based on GAs to tackle the job shop are listed next. All of these studies, it is pointed out, address the single-objective job shop.

Chapter 6 introduces the different approaches to *multiobjective* optimization. The concept of Pareto optimality and "efficient" solutions is introduced. *Niche formation* and *speciation*, two key processes occurring in nature that form the foundation of multiobjective GA methods are described in Chapter 7. The concept of *fitness sharing*, the procedure that discovers coexisting optimal solutions, is also explained.

Chapter 8 describes the non-dominated sorting GA (NSGA); Chapter 9 adapts it to sequence jobs in a three-objective flowshop. Scheduling objectives simultaneously considered here are (1) minimization of makespan, (2) minimization of mean flow time and (3) minimization of mean tardiness. Such a problem is conventionally "solved" by optimizing a weighted-sum objective.

Chapter 10 introduces ENGA, an enhancement of NSGA, which greatly increases the rate of discovery of Pareto optimal solutions. A key feature of ENGA is the *additional* non-dominated sorting step it uses in deciding which solutions would parent the next generation. Steps to write computer codes based on ENGA are described.

A statistical comparison of the relative performance of NSGA and ENGA is presented in Chapter 11. Chapters 12 and 13 solve multiobjective job and open shops. Chapter 14 is the epilog—a prognosis of using GAs in multiobjective scheduling. The Appendix provides C++ codes for a hybridized GA—to invite novices to experiment with GAs and to create their own variations.

Multiobjective flowshops, job shops and open shops each are highly relevant models in manufacturing, classroom scheduling or automotive assembly, yet for want of sound methods, they have remained almost untouched to date. This text shows how methods such as ENGA can find a bevy of Pareto optimal solutions for them. Also, it accents the value of hybridizing GAs with both solution-generating and solution-improvement methods. It envisions fundamental research into such methods greatly strengthening the growing reach of metaheuristic methods.

This work is the culmination of the effort by a large number of individuals spread over six years and two continents. I am grateful to my dear students Ankur Bhatnagar, Vince Caraffa, Stefano Ioness, Nitin Jain, K Jayaram, Subodha Kumar, Jugal Prasad, Krishan Raman, P N Rao and T D Srinivas, who "crossed" and "mutated" numerous ideas and converted these into programmable codes as our work evolved. Two youthful helpers—Shubhojit Sanyal and Paromita A Chaudhury—sought out treasured research material for us. Shiva Kumar Srinivasan read through each word of the manuscript. I have not words left to thank them all.

I am obliged to Kalyanmoy Deb, who studied with David Goldberg at the University of Alabama and subsequently introduced GAs to the students of the Indian Institute of Technology at Kanpur. I also thank Chelliah Sriskandarajah and Suresh Sethi at the University of Texas in Dallas, and Edouard Wagneur at Ecole de Mines, Nantes (France), who spent a great deal of time in discussing with me the special nuances, methods and complexity of shop scheduling. Together we hope we have turned a new page on multiobjective shop scheduling.

This work would not reach this stage without the encouragement and strong support of Jim Templeton and John Buzacott, my graduate advisors at the University of Toronto, and of Manjit Singh Kalra and Naresh Kant Batra, who guided me at the saddle points. Lastly, I acknowledge the outstanding help given by the staff of Kluwer Academic Publishers throughout the preparation of this text.

Tapan P Bagchi
bagchi@iitk.ac.in

1

SHOP SCHEDULING: AN OVERVIEW

"Getting the orders out the fastest way possible" is rarely a production manager's only objective. Customarily, he/she also attempts to minimize tardiness of the jobs, maximize the utilization of expensive capital equipment (furnaces, reactors, rolling mills, printing presses, etc.) and human resources, minimize the mean flow time of the jobs to be done, etc. Such scheduling situations are quite common and these are *multiobjective*. This book presents the methods for solving multiobjective scheduling situations modeled out of compulsion as single-objective flowshops, job shops and open shops. The methodology is meta-heuristic, one inspired by natural evolution. The method uses innovative variations of "genetic algorithms"—search methods that harness Darwin's "survival of the fittest" rule and suitable *cross-breeding*, *mutation* and *niche-formation* processes that nature is believed to have used to create the multitude of well-adapted and co-habitannt life forms on earth.

1.1 WHAT IS SCHEDULING?

Scheduling is an optimization process by which limited resources are allocated over time among parallel and sequential activities. Such situations develop routinely in factories, publishing houses, shipping, universities, hospitals, airports, etc. Solving such a problem amounts to making discrete choices such that an optimal solution is found among a finite or a countably infinite number of alternatives. Such problems are called *combinatorial optimization* problems. Typically, the task is complex, limiting the practical utility of combinatorial, mathematical programming and other analytical methods in solving scheduling problems effectively. Many scheduling problems are polynomially solvable, or *NP Hard* in that it is impossible to find an

optimal solution here without the use of an essentially enumerative algorithm. Computation times here increase exponentially with problem size.

To find exact solutions of such problems a branch-and-bound or dynamic programming algorithm is often used. Using problem-specific information sometimes reduces search space, even though the problem is still difficult to solve exactly. Such difficulty has led to the development of many *heuristic* methods and dispatching rules, many of which, however, are restricted to only special conditions under which they apply.

A heuristic method, which involves trial and error and some contemplated intuition rather than an algorithm, may at best produce an approximate solution to the NP-hard problem. Since a heuristic does not find the exact solution, there is clearly a trade-off between the computational investment in finding the solution and the quality of that solution. However, finding near-optimal solutions by approximate algorithms within reasonable time is frequently acceptable and this has become a practice in engineering, economics, management science, communications engineering and many other endeavors. Consequently, many variations of *local search* algorithms to solve NP-hard combinatorial problems have been proposed and used. When possible, local search methods are mathematically modeled to predict and improve upon their performance.

Many local search methods are inspired by processes in statistical physics, biological evolution, and more recently, neurophysiology (see Aarts and Lenstra, 1997, Chapter 7, for instance). For complex single-objective scheduling problems, the effectiveness of artificial intelligence-based constraint-respecting search methods has also been demonstrated (Zweben and Fox, 1994). The effective solution of *multiobjective* optimization problems, however, continues to evade business planners, fleet administrators, shop managers and others.

In multiobjective decision problems one desires to simultaneously optimize *more than one* performance objective, such as makespan, tardiness, mean flow time of jobs, etc. The simplest method to solve such problems is to combine the various objectives into a single objective using suitable weights and to then optimize it. This has two drawbacks. First, the weights have to be pre-specified by the decision-maker before the resulting combined objective is optimized.

Second, at best one can experimentally explore the effect of varying the weights on the solution methods. Third, the weighted sum method does not guarantee that the final solution is dominating in the Pareto optimality sense; a key character of acceptable multiobjective solutions is that they are *nondominated*, that is, each such solution is "best" at least with respect to one decision objective.

Even though the concept of Pareto optimality is at least 75 years old, good methods for finding Pareto optimal or nondominated solutions are rather few. The present text demonstrates the efficacy of a recently devised and powerful approach known as the *niche-forming genetic algorithm* in such situations. Genetic algorithms (GAs) are search methods that mimic natural evolution to rapidly find the best solution among alternatives that may be very large in number. The niche forming GA uses the notion of resource or fitness sharing. The final solutions thus produced are Pareto-optimal or "efficient," a set of solutions that are non-dominant and are thus a preferred expression of solutions to multiobjective decision problems (Zeleny, 1985; Tabucanon, 1989).

One major field of research in scheduling is the problem abstracted as "machine scheduling," the solution of which has applications in manufacturing, logistics, computer architecture design, healthcare administration, air transport management, etc. Some specific applications of machine scheduling include:

- a) Short term production planning: the determination of which and how many products to produce over time;
- b) Workforce scheduling: the determination of the number of workers and their duty cycles to meet certain labor restrictions and enterprise objectives;
- c) Timetabling: the optimum matching of participants with each other and with resources, such as student/room/examination assignments or the scheduling of sports events.

So, scheduling is ubiquitous. The first formal model for scheduling interdependent tasks was the Gantt chart (Figure 1.1) developed during World War I. This chart is a graphical representation of tasks and resources interacting over time. Critical Path Methods (CPM) followed Gantt charts and are still widely used in planning large projects and missions. The 1950's saw the growth of mathematical models in the analysis of scheduling problems. As product features multiplied and processing requirements grew in complexity

culminating in the flexible manufacturing systems (FMS), limitations of analytical methods became apparent (Pinedo, 1995). Indeed, over the past forty years scheduling has challenged some of the best analytical minds in operations research. Since 1970, several good heuristics have been also proposed.

The reason for this enduring attention to scheduling is its importance in production cost control and customer satisfaction, and the amount of time that is routinely spent by firms *only* on the scheduling function (Baker, 1974; Magazine, 1990). This function is typically at the operational level and it assumes that the planning phases of which tasks are to be done "today" or "this week" and what resources are available to complete them have been completed.

Still, only a few results have made a major impact on the *practice* of scheduling so far (McKay et al., 1988). This is due to the very difficult nature of the general problem and the myriad constraints and locale of its applications. Broadly stated, scheduling, the process by which limited resources must be efficiently allocated over time among parallel and sequential activities involves the resolution of two types of decisions:

- (a) *When* do we perform each of a given set of tasks?
- (b) *Which* resources will be assigned to perform each of these tasks?

The next section outlines the general scheduling problem.

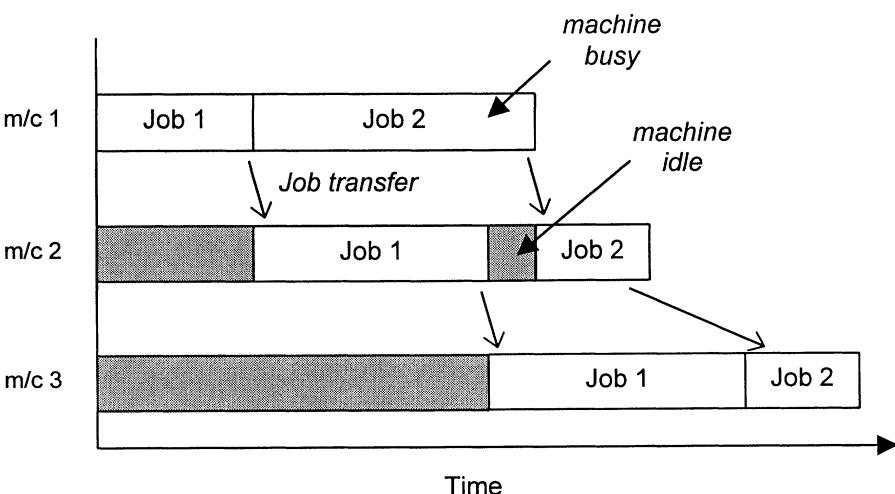


FIGURE 1.1 THE GANTT CHART

1.2 MACHINE SCHEDULING PRELIMINARIES

In general, a "job" represents a distinct and identifiable task or set of activities with a distinct objective, a distinct beginning and a distinct end point. In this text we restrict our attention to *deterministic* machine scheduling, situations in which the data about the jobs and the machines that define a problem are known with certainty in advance. A job may consist of several processing steps, done on several different machines or processing centers, determined by technological requirements. Scheduling decisions must resolve the *when* and *which* in order that performance measures such as tardiness, work-in-process inventory, and makespan are minimized. The aspects of the problem that affect these decisions include

- Machine configuration
- Job characteristics
- Objective functions

The machine configuration or the job processing environment may be categorized into *single* machine problems and *multiple* machine problems (Gupta and Kyaparisis, 1987). Single machine problems usually require the jobs to be optimally sequenced. The problems in the multiple machine category are *parallel-machine* scheduling problems (Cheng and Sen, 1990), *flowshops* (these involve several machines laid out in series in the shop) (Dudek et al., 1992; Morton and Pentico, 1993), *job shops* (these involve several machines without the series structure) (Muth and Thompson, 1963; Conway, Maxwell and Miller, 1967; Baker, 1974; Coffman, 1976; French, 1982; Baker and Scudder, 1990; Blazewicz, Ecker, Smith and Weglarz, 1994; Morton and Pentico, 1993; Sannomiya and Iima, 1996) and *open shops*. Variations of these basic models have appeared, some important ones coming from FMS (Wellman and Gemmill, 1995; Jain and Elmaraghy, 1997).

A flowshop is characterized by unidirectional flow of work with a variety of jobs all being processed sequentially in the same order, in a one-pass manner. Typically, each job is different but every job follows the same one-pass routing through the processing stages (we call these stages "machines"). We assume unlimited storage between machines in a flowshop.

The challenge in scheduling the flowshop is to determine the optimum sequence in which the jobs should be processed in order that

one or more performance measure, such as the total time to complete all the jobs, the average mean flow time, or the mean tardiness of jobs from their committed dates is minimized.

A job shop, on the other hand, involves processing of the jobs on several machines without any "series" routing structure. The standard definition of a job shop is as follows. The facility produces goods according to pre-specified process plans, under several domain-dependent and commonsense constraints (Uckun, Bagchi, Kawamura and Miyabe, 1993). A job shop can produce several parts when each part may have one or more process plans. Each process plan consists of a sequence of operations. Each such operation requires machines, personnel, raw material, etc. and it needs a certain time on a certain machine. The object of optimally scheduling the operations is similar to that for the flowshop: we wish to minimize makespan, mean flow time, mean tardiness, etc.

Open shops are similar to job shops with the exception that there is no *a priori* order of the operations to be done *within* a single job (Pinedo, 1995). Customized (made-to-order) automotive assembly and classroom scheduling situations represent open shops. The job-processing environment frequently emerges as the factor of key significance in scheduling. It often determines the degree to which we are able to reach the optimal solution of the problem at hand.

Job characteristics typically impose *constraints* on the solutions. Constraints can make certain solutions unacceptable or infeasible. Constraints may include conditions such as precedence relations amongst the jobs, job priorities, job release dates, job due dates, set-up times of machines and pre-emption capabilities. McKay et al. (1988) have identified over 600 types of constraints in manufacturing.

Even though many applications involve the determination of a schedule that is simply *feasible*, given the scarce resources, most models use some economic measures of performance to enable us to compare the "quality" of different schedules. These performance objectives typically represent some surrogate of throughput, customer satisfaction, or costs. The production planner commonly focuses on a single objective or criterion at a time although solutions are sought for multi-criteria scheduling problems also (Daniels, 1990; Dudek, Panwalker and Smith, 1992).

We often find that models representing scheduling problems become increasingly difficult to solve as the environment, job characteristics and objective functions become more and more realistic, incorporating in them the realities of the factory particulars, product features, technologies, etc. Many machine scheduling problems are NP-hard, as already mentioned. Also adding to such difficulty is the imprecision or uncertainty inherent in the data in some problems. Processing times, for instance, may not be exactly known. Most textbook models and their solutions assume that the data are deterministic. By contrast, there exist only a limited number of solutions for scheduling problems that involve probabilistic processing times, interruptions, etc. (Pinedo and Schrage, 1982).

1.3 INTELLIGENT SOLUTIONS TO COMPLEX PROBLEMS

Intelligence is defined as the faculty of quickness in understanding and comprehension, linked to the ability of solving problems. In the last forty years much effort in computer science has been directed toward making computers to do "intelligent" things (such as doing the requisite "thinking" to solve complex problems) that at the moment people do better. Many philosophical issues have been tabled and addressed in the process. On the tangible plane, the field of "artificial intelligence" or *AI* has been created.

A key mission of *AI* has been to tackle problems that are very hard or too large to solve by direct, deductive methods. The examples of such problems are playing chess competitively or scheduling an assortment of jobs in a machine shop to minimize makespan, or the total idle time of all machines.

AI research has now demonstrated that intelligence requires more than the ability to reason. It has shown that intelligence requires a great deal of knowledge about the problem domain. Work on *AI* has also created a variety of methods for encoding knowledge in computer systems. These include predicate logic, production rules, semantic networks, frames, scripts and many other schemes. Broadly speaking, however, approaches to solve hard problems seem to take three important and (often a combination of) distinct avenues.

The first is *search*, the "find solutions and examine them thoroughly" strategy to solve problems for whom a direct approach is not available. Search also often provides a framework in which any direct technique that is available can be embedded. The second approach is the use of *knowledge*, the *range* of information available to the problem solver. Knowledge assists in the solution of problems by exploiting the structure of the objects that are involved. The third approach is to do *abstraction*. In this approach we separate out the important features of the problem from the unimportant features that drown the *process* (the mechanism of what causes what and how) underlying the problem.

If successful abstraction is possible, the method of solving the problem may be reduced to stepping though a finite series of steps, or an algorithm. If no efficient algorithm can be designed, one may settle with a less-than-perfect solution approach.

Experience shows that solution approaches that exploit search, knowledge or abstraction to tackle hard problems are reasonably robust. They are able to find solutions in each of a wide variety of input conditions. Also, such methods are effective where more direct solution methods break down, often due to the vastness of the solution space.

Many real life optimization problems such as playing chess or scheduling a machine shop can be defined formally, so that one is able to conceptualize "rules" that when applied together can ensure the problem's solution. Depending, however, upon the number of possible solutions, the number of rules that we require to completely describe all transitions (or paths) to reach the optimum solution can become immensely large. A good problem solving strategy, therefore, would be one that would allow us to *move* rapidly through a very large search space. Also, a good solution strategy would be systematic in the sense that it would *progress* rather than re-visit the earlier developed solutions too many times. Solving a scheduling problem by integer programming is one such strategy.

However, many hard problems may not allow us to search through the myriad solutions systematically and with mobility—within reasonable time. In such cases we have to compromise on the requirements of mobility and systematicity—out of necessity. Here we construct a *heuristic*, a technique that allows us to discover

solutions that point us in interesting directions (i.e., toward the optimum solution), though they may be "bad" to the extent that they may miss the true optimum. A good heuristic method can reach us good (though possibly non-optimal) solutions to hard problems in a reasonable amount of time. Indeed, without heuristics, in conducting a search we would often become ensnared in a combinatorial explosion.

While we mention heuristic methods, it is of value to recall the role played by knowledge in solving a problem. Knowledge is helpful in two ways. There are some problems for which a lot of knowledge is required only to constrain the search for a solution, to keep the search effort confined within some reasonable limit of the solution space. On the other hand, there are problems for which a lot of knowledge is required only to permit us to recognize a solution.

Thus, there are many problems that are too complex to be solved by direct techniques. In other words, the optimum solution to these problems cannot be found easily. Rather, these problems may be attacked by heuristic methods assisted by whatever direct techniques are available at hand to guide the search. These methods are frequently describable independent of the particular optimization task or problem domain. Some heuristic methods, however, are such that their efficacy is highly dependent on the way they exploit domain-specific knowledge. This may be sometimes troublesome, for heuristic methods are in themselves unable to overcome the combinatorial explosion to which search processes are so vulnerable.

Perhaps the generate-and-test search strategy is the simplest of all search methods. In it we generate a possible solution. To see if it is actually a solution we compare its evaluation to the set of acceptable goal sets. If a solution is found, we quit. Otherwise we generate another solution. When done systematically, this procedure will find the solution eventually. For simple problems, generate-and-test is often a reasonable approach. For large problems the solution generation step may be randomized. However, then there is no guarantee that a solution will ever be found.

Many local search algorithms are generally applicable and flexible. They require a cost function, a neighborhood function, and an efficient method for exploring the neighborhood. Nevertheless, local search often produces poor local optima. To remedy this the scope of local

search is frequently broadened. One method is the *multistart* approach. In it a simple local search algorithm is run a number of times using different start solutions. The multistart approach, applied alone however, exhibits limited success for both the travelling salesman problem—another NP hard problem, and the job shop scheduling problem (Aarts and Lenstra, 1997).

A variety of neighborhood search methods have been now created including hill climbing, simulated annealing (Connolly, 1992), best first and A*. These methods offer heuristic refinements to generate-and-test (Rich and Knight, 1983). Constraint satisfaction is another search procedure that operates in a space of constraint sets rather than in the solution set space. Yet one different paradigm views *learning* (occurring as the solution method iterates) itself as a problem solving process.

Several such approaches take advantage of search strategies in which even cost-deteriorating neighbors are accepted. While a learning "machine" that begins to solve problems with solution traces to example problems and finds new heuristics for solving new problems efficiently remains perhaps still some distance away, several strategies use learning already. The method known as *simulated annealing* uses an analogy with the physical process of annealing, in which a pure lattice structure of a solid is made by heating up the solid in a heat bath until it melts, then cooling it down slowly until it solidifies into a low-energy state. As designed, simulated annealing is a randomized neighborhood search algorithm and it has been successfully applied to solve many single objective scheduling problems. Simulated annealing has also been modeled mathematically and its convergence is typically exponential (Aarts and Lenstra, 1997). Another method, called *tabu search*, combines deterministic iterative improvement with the possibility of accepting cost-increasing solutions occasionally—to direct the search away from local minima (Glover and Laguna, 1997). Yet another method in which learning occurs through a solution selection process is the *genetic algorithm* (GA). Even *artificial neural nets* have been used in optimization, including job shop scheduling (Aarts and Lenstra, 1997). The present text probes the exploits of the GA in tackling difficult scheduling tasks.

GAs are *meta-heuristic* methods or heuristic methods *about* heuristics used to solve problems. The invention of the GA and the related methods called evolutionary algorithms has been actually the

discovery of a powerful and new learning algorithm, even if a mathematical proof for its convergence is still wanting. Original proposals to mimic natural evolution go back to 1960, including the work of Bremermann, Roghson and Salaff (1966). The independent construction of the popular version called the GA (Holland, 1975) was inspired by natural selection and adaptation. It should be noted that here one should be *inspired* by nature, rather than copy nature blindly, for nature, it is said, is not attempting optimization (Smith, 1993). The GA methodology seeks to achieve small local climbs (adaptations) without ever requiring knowledge of gradient information about the objective function being optimized. The GA begins with an initial population of random solutions. But soon it "learns" and it begins to exploit interesting regions of the solution space using "fitness" of the solutions at hand to generate new solutions. Also, a GA search cleverly parallelizes enterprise across many different hyperplanes at the intersection of which the global optima are expected to lie.

1.4 SCHEDULING TECHNIQUES: ANALYTICAL, HEURISTIC AND META-HEURISTIC

A typical scheduling problem comprises several concurrent and often conflicting goals and a multitude of resources available to satisfy those goals. The combination of several goals and resources, it is to be noted, may result in an exponentially growing number of possible solutions. In such cases it becomes difficult or even impossible to find exact solutions in reasonable time.

As a rule, scheduling problem formulations engage some economic performance criteria. It is therefore natural that we *optimize* these problems. Many scheduling formulations seek combinatorial solutions. However, the number of feasible schedules in combinatorial optimization grows very quickly with the number of jobs, number of machines, number of processing steps, etc., making almost all problems of practical significance difficult to solve optimally. For instance, one may consider the simplest scheduling problem with n jobs to be performed in one-pass manner on a single machine, with no additional constraints, when the objective is the minimization of the tardiness of the jobs from their due dates. Here there would be $n!$ distinct job sequences possible that must each be evaluated. Even for a relatively small problem when say $n = 100$,

optimization by constructing each possible job sequence and then evaluating it (the method being known as *enumeration*) would be all but impossible. An n -machine, m -job problem has $(m!)^n$ possible schedules, a phenomenon called combinatorial explosion. Combinatorial explosion thus is a critical difficulty in solving scheduling problems. The other difficulty in scheduling is the diversity of conflicting constraints posed by due dates, cost limits, machines, order characteristics, etc. (Zweben and Fox, 1994).

Perhaps the most celebrated analytical result in shop scheduling for which a theoretical proof exists for its optimality is Johnson's rule (Johnson, 1954). Johnson's rule optimally solves the 2-machine flowshop makespan minimization problem and for special situations it may be extended to the 3-machine flowshop. Beyond Johnson's rule, few general and simple-to-apply results exist (Pinedo, 1995).

Based on the complexity of the algorithm used to solve optimization problems, all problems may be classified into two classes, called P and NP in the literature. Class P consists of problems for which the execution time of the solution algorithms grows polynomially with the size of the problem. Thus, a problem of size m (in this class) would be solvable in time proportional to m^k , when k is an exponent. The time taken to solve a problem belonging to the NP class grows exponentially, thus this time would grow in proportion t^m , when t is some constant. In practice, algorithms for which the execution time grows polynomially are preferred. However, a widely held conjecture of modern mathematics is that there are problems in NP class for which algorithms with polynomial time complexity will never be found (French, 1982). These problems are classified as *NP-hard* problems. Unfortunately, most of the practical scheduling problems belong to the *NP-hard* class (Rinnooy Kan, 1976; Lenstra, Rinnooy Kan and Brucker, 1977).

Beginning about 1960, traditional optimization techniques have been used successfully to solve a few such problems. There are instances in which mathematical programming and intelligent enumeration methods based on branch and bound or dynamic programming have reduced the analyst's computational burden in reaching optimal solutions. Most often, however, the size or the complexity makes it necessary for one to resort to *heuristic* techniques. Such techniques can obtain a "solution" (without the guarantee of optimality) to a large problem—with limited computational effort. In many such cases the

computational requirements are predictable for problems of a given size. However, the primary drawback of heuristic methods is that they do not guarantee providing an *optimal* solution to the original problem.

Still, heuristic job or machine scheduling methods can be very useful when no other method works. For instance, the guided local search method by Balas and Vazacopoulos (1994) is claimed to solve job shop problems involving 100 jobs and 20 machines within 0.5% of optimality in a few minutes. Broadly, heuristic methods are of two types: (1) Those that limit the space of search by only considering schedules that meet some specified *criteria*. (2) Those that search in a limited *neighborhood* of some known feasible schedule to improve the solution at hand. Extensive recent research on the evaluation of the performance of heuristics on a variety of scheduling problems has given some additional promise to their validity. However, we observe that most such work considers often only a heuristic's *worst-case* performance, which is not necessarily a true measure of goodness of the utility of a given heuristic (Magazine, 1990).

In problems involving stochastic data, combinatorial techniques may be sometimes used. The simpler problems often take advantage of queuing theory results but, by and large, Monte Carlo simulation (Law and Kelton, 1991) becomes the technique of frequent choice. Often the complicated environment or simply the highly combinatorial nature of the problem is enough to render simulation the only viable alternative for an attempt at the solution.

More recent advances in *meta-heuristic search* methods—methods that help conduct directed "intelligent" search of the potentially very large solution space—have brought new possibilities to our ability to search for efficient and economic schedules. As mentioned, these methods now include techniques known as genetic algorithms (Holland, 1975; Dowsland, 1996), tabu search (Glover and Laguna, 1997), threshold acceptance, simulated annealing (Morton and Pentico, 1993) and artificial neural nets (Aarts and Lenstra, 1997).

GAs discover superior solutions to global optimization problems *adaptively* akin to the evolution of organisms in the natural world, looking for small, local improvements rather than big jumps in solution quality. While most stochastic search methods operate on a

single solution to the problem at hand, the GA operates on a *population* of solutions.

To use GA, however, you must first *encode* the solutions to your problem in a *structure*. The procedure then applies *crossover* and *mutation* (processes inspired by natural evolution) to these structures—the individuals in the population—to generate new individuals (solutions). The GA uses various selection criteria so that it picks the best individuals for *mating* so as to produce superior solutions by combining parts of parent solutions intelligently. The objective function of the problem being solved determines how "good" each individual is.

Theoretical work on multi-criteria scheduling began in the 1970s. Heck and Roberts (1972) presented a method to consider a secondary criterion. Emmons (1975) solved an n -job one-machine problem with weighted tardiness of the jobs and their completion time as two separate criteria. Van Wassenhove and Gelders (1980) solved the same problem using holding cost and maximum tardiness as the criteria. Van Wassenhove and Baker (1982) developed the efficient frontier (the Pareto front) for a single machine shop, treating completion penalties and resource allocation as objectives. Dileepan and Sen (1988) studied the bi-criteria single machine scheduling problem.

The present text overviews the GA approach as applied to shop scheduling. Subsequently, it addresses the task of finding Pareto optimal or efficient solutions to *multiobjective* flow, job and open shop scheduling problems—by genetic algorithms.

The method presented is extendible to multi-criteria FMS scheduling also. The procedure uses variations and significant enhancements of the Nondominated Sorting Genetic Algorithm (NSGA) (Srinivas and Deb, 1995), a meta-heuristic method recently proposed, which produces Pareto-optimal solutions to numerical multiobjective problems. One such enhancement introduced in this text is the *Elitist Nondominated Sorting Genetic Algorithm* (ENGA). The object in such methods is singular: solve a variety of multiobjective optimization problems, *and* do it efficiently. The final solutions are all Pareto-optimal or "efficient."

1.5 OUTLINE OF THIS TEXT

Genetic algorithms are the mainstay of the methods presented in this text. A beginner's introduction to genetic algorithms is provided in Chapter 2. Chapter 3 addresses an important aspect of the practical use of GAs—it is the effective *parameterization* of a GA to make sure that the algorithm will converge to near-optimal solution in the shortest span of time. A discussion of the flowshop scheduling problem is given in Chapter 4, in which we also review the important heuristic rules available in the literature that solve the *single-objective* flowshop problem. A summary of some recent methods for using GAs to find the optimal job sequence for a flowshop is also given including a new scheme for smart hybridization that uses Lamarckism. With one or two exceptions, however, all these methods tackle only the *single-objective* flowshop.

The job shop scheduling problem, a problem in which the routing of the different jobs through various machines is not identical, is described in Chapter 5. Indeed, the job shop is widely found in industry and hence it continues to occupy the time and attention of a truly large number of shop planners, operations research experts and industrial engineers. The important known results available in the literature on job shop problems are summarized in Chapter 5, including mathematical programming formulations and some noteworthy heuristic rules. Approaches based on genetic algorithms to tackle the job shop are next summarized. Again, as with the flowshop, almost all these studies address only the *single-objective* job shop.

Chapter 6 introduces the different methods of approaching *multiobjective* or multi-criteria decision problems. The concept of Pareto optimality is introduced and the different methods for finding Pareto optimal solutions are outlined. The chapter concludes with the mention of a recently published GA method known as non-dominated sorting GA (NSGA) (Srinivas and Deb, 1995). NSGA uses the notion of niche forming by resource sharing as it occurs in nature to allow diverse varieties of species to coexist in the ecosystem. NSGA produces Pareto-optimal solutions to multiobjective optimization problems.

The genesis of GAs was an insightful observation by John Holland (1975) that some aspects of natural evolution could be cast into useful

algorithms to seek out solutions to the more difficult optimization problems. Holland based GAs on an important basic process known as adaptation. Chapter 7 describes two phenomena closely related to evolution, known as *niche formation* and *speciation*. Niches are resource-sharing behavior patterns that develop when organisms compete with each other for limited resources or when they attempt to survive in unfavorable environmental conditions. Speciation is the process by which new and stable species evolve in natures. Subsequently, in Chapter 8 we describe NSGA, the multiobjective GA that uses the notions of niche formation and speciation to discover Pareto optimal solutions to multiobjective problems.

A difficult numerical optimization problem—a bi-criteria robust design problem—is solved by NSGA in Chapter 8 to illustrate the steps in NSGA. The final designs produced here are all Pareto-optimal.

The multiobjective flowshop problem is tackled in Chapter 9 using NSGA. Chapter 9 also includes a review of earlier published methods to tackle the multiobjective flowshop. Chapter 10 introduces an enhancement to NSGA, one that incorporates *elitism* (the preservation of good GA solutions to the next generation). Dubbed ENGA (the elitist nondominated sorting genetic algorithm), this algorithm significantly improves the speed at which Pareto-optimal solutions are discovered. The distinctive feature of ENGA is the *additional* nondominated sorting step it uses after reproduction in deciding which solutions would constitute the next generation of solutions. Chapter 10 uses the design-of-experiments approach to parameterize ENGA optimally to ensure its rapid convergence. In order to facilitate the computer implementation of ENGA in C++ or in a similar language, details of sample codes are provided in Chapter 10, using the multiobjective flowshop as an illustration.

Several tri-criteria flowshop problems are solved in Chapter 11, by NSGA as well as by ENGA, to compare the performance of these two different multiobjective genetic algorithms. The results are compared statistically. It is concluded that ENGA forms a more efficient method to develop Pareto-optimal solutions.

The three-objective job shop problem is examined in Chapter 12, again using both NSGA and ENGA. The statistical tests of the results indicate that ENGA is a more efficient approach.

The open shop scheduling problem is studied in Chapter 13, using a *bi-objective* open shop scheduling problem as a test bed. Solving such a problem is highly relevant in situations such as classroom scheduling or automotive assembly, yet it is one that has remained unsolved to date. In reality, in structure, the open shop represents a much wider search space than the job shop, as there is no *a priori* ordering of operations within the jobs. Here again, ENGA seems to be more efficient.

Chapter 14 provides a prognosis and some reflections on using genetic algorithms to tackle multiobjective shop scheduling and other similar problems.

The Appendix contains the Boreland Turbo C++® code of a software developed by Jain (1999) that solves the single objective flowshop by conventional heuristic methods, the standard Darwinian GA and the hybridized GA incorporating the Ho-Chang flowshop heuristic. If you have not wetted your toes in GA yet, you may use these codes for experimentation.

While the methods as presented in this book address only "static" flowshop, job shop and open shop problems, solutions to multi-criteria scheduling problems even for the static case are not easy to develop. The method based on GA appears to be a reasonable approach when no other methods would seem satisfactory.

NSGA or ENGA can solve the multi-criteria static open shop. The open shop is often used to model classroom scheduling. Hence, the method shown in Chapter 13 may be adapted relatively easily to tackle multi-criteria classroom scheduling, a problem as common as the number of universities and colleges that abound. Such problems are conventionally solved as single-objective problems or by objective weighting, if at all (Carter, 1997).

This text has used GA as the solution paradigm to seek Pareto optimal solutions because the GA is population-based. It is very possible to combine "single-solution" type methods to find Pareto solutions (such as the ε -method (Section 6.6.1)) with simulated annealing or tabu search. Incorporation of *learning*, however, may bring bigger dividends when the objective is efficiency. Such interesting possibilities are yet to be extensively explored.

This text has assumed *Pareto optimality* as the basis for rational choice (Zeleny, 1985). This was deliberate because such a rationale does not require prior specification of the decision maker's preferences, which may be easier to express, obtain or develop once the nondominant solutions are at hand. A weighted-sum formulation of the multiobjective problem is actually a simpler problem for the GA because a suitably constructed simple GA (SGA, Section 2.6) alone can solve that problem (Section 4.7). The weighted-sum solution, however, will not be nondominated, in general.

This text has illustrated the utility of two natural phenomena (namely niche formation and speciation) to enhance the capability of the simple GA in shop scheduling. These two innovations in GA, as shown in this text, are of high value in solving multi-criteria sequencing, scheduling and assignment problems. However, we expect that a deeper understanding of nature's other processes (see, for instance, Wilmer, 1990, Dawkins, 1996 and Russell, 1998) and clever schemes to incorporate learning may evolve more powerful tools and methodologies. Some of these possibilities have been hinted to by Goldberg (1989, p. 190) and others including Chakraborti and Sastry (1998). Such developments, we foresee, may lead to the construction of GAs that may even *evolve* smart heuristics for problems in FMS and ERP and in other complex decision domains.

Scheduling is NP-hard. As far as solving multiobjective combinatorial problems such as those appearing in hospital staff/operations scheduling, timetabling and production scheduling by GA goes, we expect hybridizing, fitness sharing and domain-specific knowledge incorporation to produce the most effective as well as efficient codes.

2

WHAT ARE GENETIC ALGORITHMS?

This chapter describes "genetic algorithms" (commonly called "GAs")—a set of global search and optimization methods that is fast gaining popularity in solving complex engineering optimization problems with a large search space. GAs belong to the class of meta-heuristic methods known as evolutionary algorithms that model natural evolution processes. GAs evolve solutions to global optimization problems *adaptively*. Adaptation is a term that we have borrowed from biology implying those structural or behavioral changes in an organism that evolve for their current (or local) utility. Two other meta-heuristic methods that are also used frequently in global search are simulated annealing and tabu search. GAs systematically evolve a *population* of candidate solutions—with the objective of reaching the "best" solution—by using evolutionary computational processes inspired by genetic variation and natural selection.

2.1 EVOLUTIONARY COMPUTATION AND BIOLOGY

In the 1960s the idea of solving optimization problems by using notions of evolution of organisms occurring by natural adaptation was first conceived. The initial work here includes Bremermann, Roghson and Salaff (1966). In 1859 Charles Darwin had put forth a plausible explanation of how natural genetic variation and the principle of *natural selection* together result in evolution. Evolution is the series of slow changes that occur as populations of organisms adapt to their changing surroundings. Darwin explained that when resources are limited or the environment changes, organisms undergo a struggle for existence. Subsequently natural selection (a "survival of the fittest" struggle) occurs. Organisms that survive such selection are

allowed to procreate and pass on their special survival capabilities to their offspring. Thus, according to Darwin, a "better fitted" generation evolves.

While designers of evolutionary algorithms should be inspired by nature, we need to point out that one should not intend here to *copy* nature. An algorithm would be good if it solves difficult problems, not because it uses a "pure" natural principle. Also, an algorithm cannot be bad because the logic it uses cannot be found in nature. The key reason for this assertion is that nature may inspire, but it seldom seeks perfection itself. Nature, though it does an excellent job in adaptation, does not appear to aim at optimizing some objective.

Recently, recasting optimization problems using Darwin's ideas has led to solutions to many difficult tasks including the optimum design of airfoils, bridges, oil transport systems and special chemical molecules, among many other objects and processes. The other evolution-inspired ideas for optimization include EVOP by Box (1957) to improve manufacturing processes and the simulation of biological evolution to improve machine learning.

John Holland invented GAs (as we know them today) in the 1960s during his attempts to formally model and explain adaptation as observed in nature. GAs were developed further by Holland himself and by his graduate students. Holland's GA (1975) is a method for moving from one population of "chromosomes" (solutions represented by strings of binary bits) to a new population by using natural selection together with the genetic processes of crossover, mutation and inversion. This population-based approach was a major revolution in optimization by search. Holland was also the first to attempt to put computational evolution on a theoretical foundation, based on his notion of "schemas."

Searching (Section 1.3) through a large number of possibilities for optimal solutions is a common enough problem in engineering, medicine, economics and a number of other fields. Whenever possible, what is helpful here is some sort of parallelism (e.g., many solutions being evaluated simultaneously) and an intelligent strategy for choosing the next set of solutions to evaluate. Holland used analogies from biology to design such a procedure. Though biological terms are used in describing the GA in the spirit of analogy, the

concepts involved in GA become easily understood when put in terms of natural genetics. Some of these terms are as follows.

Evolution, as we saw earlier, is the gradual process by which the present diversity of plant and animal life arose from the earliest and most primitive organisms, which by fossil records is believed to have been continuing for at least the past 3 billion years. Until middle 1800's it was generally believed that each species was divinely created and fixed in its form throughout its existence. French biologist Jean-Baptist de Lamarck (1744-1829) perhaps provided the first departure from it. He published a theory in 1809 to explain how one species could have evolved into another (Smith, 1993). Lamarck suggested that changes in an individual are acquired during its lifetime, chiefly by increased use or disuse of organs in response to "a need that continues to make itself felt" and that these changes are inherited by the organism's offspring. Trofim Lysenko (1898-1976), who dismissed classical genetics, denied the existence of genes and held that variability of organisms was produced solely by environmental changes, adopted a similar view.

But it was not until Charles Darwin (1809-82) wrote *On the Origin of Species* (Darwin, 1860) that the "divine creation" theory was directly challenged. Darwin proposed a feasible mechanism for evolution and backed it up with evidence from fossil records and studies of comparative anatomy and embryology. The modern version of "Darwinism" incorporates discoveries in genetics (including Mendel's 1866 postulates about heredity) made since Darwin's time. Neo-Darwinism today is probably the most acceptable theory of species evolution. Mendel had suggested on the basis of extensive experiments that he conducted, that individual characteristics were determined by inherited "factors." Later, when microscopes revealed details of cell structure the behavior of Mendel's factors could be related to the behavior of chromosomes during cell division to form reproductive cells in organisms (Russell, 1998). Current explanations of the relationships and evolution of groups above the species level, however, still remain controversial.

Organisms as we find them today in their natural habitats are *well adapted* to their modes of life, but according to Darwin, this good fit is an indirect result of a process that he termed natural selection. Darwinism postulates that present day species have evolved from

simpler ancestral types by the process of "natural selection" acting on the variability found within populations.

When first published, Darwinism caused a furor because it suggested that species are not immutable nor were they specially created—a view directly opposed to the doctrine of divine creation. The current theory of the process of evolution, known as neo-Darwinism, combines evidence from classical genetics with the Darwinian theory (natural evolution). It makes use of modern knowledge of *genes* and *chromosomes* to explain the source of genetic variation upon which natural selection works. Chromosomes are threadlike structures of proteins, deoxyribonucleic acid (DNA) and ribonucleic acid (RNA) present in the nucleus of plant and animal cells. Chromosomes carry the genes (a unit of heredity composed of DNA) that determine the individual characteristics of an organism.

Darwin supported his assertion by first citing three basic facts (Gould, 1977):

1. All organisms tend to produce more offspring than can possibly survive.
2. Offspring vary among themselves, and are not carbon copies on an immutable type.
3. At least some of this variation is passed down by inheritance to future generations. (Darwin did not know the mechanism of heredity, for Mendel's principle did not get acceptance until early in the 20th century.)

The inference that emerges from these three facts, according to Darwin, is the principle of natural selection: If many offspring must die (for not all can be accommodated in nature's limited ecosystem), and individuals in all species vary among themselves, then on the basis of statistical average, survivors will tend to be those individuals with variations that are fortuitously best suited to the changing local environment. Since heredity exists, the offspring of survivors will tend to resemble their successful parents. The accumulation of those favorable variants through time will produce evolutionary change.

Darwin's theory, however, is more complex than Lamarckism. It requires two separate processes rather than a single force. Both theories require adaptation, the notion that organisms respond to changing environments by evolving a form, function, or behavior better suited to these new conditions. Thus, in both theories, as noted

by Gould (1993), information from the environment must be transmitted to organisms. In Lamarckism, the transfer is direct. An organism (say, the giraffe) perceives the environmental change (e.g., it finds that the leaves it wants are high on tall tree branches), responds in the "right" way (by stretching its neck each time it reaches for those leaves) and passes its appropriate reaction directly to its offspring. Similarly, if hairy coats are better, animals perceive the need, grow them, and pass the potential to offspring. Thus, variation is directed automatically toward adaptation and no second force like natural selection is needed. Consequently, Lamarckism holds that genetic variation originates *preferentially* in adaptive directions.

Darwinians, on the other hand, speak of genetic variation, the first step, as "random." Here by randomness we simply mean that variation occurs with no preferred orientation in adaptive directions. If temperatures are dropping and a hairier coat would aid survival, genetic variation for greater hairiness does not begin to arise with increased frequency. Selection, the second step, works upon unoriented variation and changes a population by conferring greater reproductive success upon advantageous variants.

Therefore, a few key points, as follows, must also be noted about the "Darwinian" version of evolution.

- (1) Darwinian evolution aims only at *local adaptation* and it works by the indirect and inefficient (slow) mechanism of natural selection. We have no evidence that the modal form of human bodies or brains has changed at all in the past 100,000 years.
- (2) Inheritance that is suitable for evolution is *genetic*. The giraffe stretching its neck upward or the blacksmith developing a strong right arm cannot pass on these advantageous "acquired characters" to offspring, for these acquired characters that reshape their physical or observable appearance or *phenotype* do not alter the genetic material that will build the next generation. The phenotype of an organism is determined by its genes, the dominance relationships between the alternative forms one gene can take, and by the interaction of the genes with the environment.
- (3) Darwinian evolution at the species level and above is a continuous and irreversible proliferation. Species are defined as distinct members of a population that are unable to reproduce with members of any other species. Each species branches off

from ancestral stocks—discrete populations inhabiting a definite geographical area and it establishes its uniqueness by evolving a genetic program sufficiently distinct that members of the species breed with each other but not with members of other species. This phenomenon is called *speciation* and we discuss it in Chapter 7.

Once a species (defined by its inability to reproduce with members of any other species) becomes separate from an ancestral line, it remains distinct forever. Thus, natural evolution is a process of constant separation and distinction. As Gould (1977) puts it, the process simply allows individuals to struggle in an unfettered way for personal gain. In this struggle, the inefficient are weeded out and the best balance each other to form an equilibrium (consisting of stable multiple species) to everyone's benefit.

The last point, (4), to be noted is that natural evolution as suggested by Darwin is a copiously branching bush with innumerable present outcomes ranging from simple parasitic bacteria to *homo sapiens* and butterflies, not a highway or ladder to *one* summit. It would be educative to read Gould's interpretations and explanations at this point. Gould shows why nature has not aimed at progress or perfection, but rather at local adaptation. Changes achieved in nature (here Gould quotes Darwin) aim at progress at a broad, overall average level to aid "adaptation chasing local environments" (Gould, 1993).

A great deal of understanding has also been gained, particularly in the past 50 years, about what life is made up of and how it moves on. We know today that life is not merely cells but of complex, ordered molecules, the nucleic acids, and a structured body of proteins. In the following pages we recapitulate some aspects of natural evolution and its processes but we restrict our discussion to aspects that have had a bearing on the design of genetic algorithms. For details that have their mooring in biology and in natural history, and their fascinating interpretations, we refer you to Young (1971) , Gould (1996) and Russell (1998).

Why does evolution occur? A key reason for natural evolution as given by Darwin is that growth of population if unchecked will outrun any increase in food supply in nature. Therefore, a struggle for existence must arise, leading by natural selection to the survival of

the fittest. As Gould has noted, Darwin was inspired to think along these lines by the works of Adam Smith (Smith, 1937), who proposed the doctrine of *laissez-faire* economics. Darwin suggested that since heredity exists (we can observe easily that children resemble their parents), the offspring of survivors will tend to resemble their successful kin. The accumulation of these *favorable* variants in the population through time will produce evolution.

While experimenting with sweetpeas, beginning in 1856, Gregor Mendel (1822-84) hinted that the inheritance of parental qualities is genetic. Subsequently, biologists learned a great deal more about heredity. As we now understand, the grist of inheritance or genetic information in an organism is held in the nuclei of the cells contained in the body of the organism. It is held in deoxyribose nucleic acid (DNA) and ribonucleic acid (RNA) molecules contained in the cell nuclei. All living cells belonging to organisms contain one or more *chromosomes* (strings of DNA) that serve as the "blueprints" or the determinants of the organism's characteristics and its behavior. Sections of this DNA string, called *genes*, are the functional blocks. It is now well established that genes hold the traits that the organism manifests. Each gene encodes a particular property or trait of the organism such as eye color or height in humans or the type of flower a plant produces. The different possible traits of a gene (called *alleles*) determine the organism's different possible traits. For instance, blue eyes or brown eyes correspond to the different alleles of a particular human gene. Furthermore, each gene is located at a particular *locus* (position) on the chromosome. Thus, just as words are made of different sequences of letters, so the genetic message is made up of different sequences of genetic letters called nucleotides. The message written in the nucleotides in turn controls the manufacture of specific proteins in the cell of an organism.

Many organisms have multiple chromosomes in each cell. Human cell nuclei each have a set of two (i.e. a *pair* of) chromosomes. The complete collection of all the genes in a single set of chromosomes (the complete collection of genetic material) is called the organism's *genome*. The term *genotype* implies the particular set of genes contained in a genome. Thus two individuals having identical genomes are said to have the same genotype. Genotypes in turn give rise to phenotypes, the organism's particular physical and observable characteristics such as height, hair color and behavior such as intelligence.

Organisms containing chromosomes in pairs are called *diploid*. Most sexually reproducing organisms are diploid. Normally, humans have 23 pairs of chromosomes in each non-germ cell in the body. By contrast, organisms whose chromosomes are unpaired are called *haploid*.

Growth or multiplication of an organism occurs by cell division, the formation of two daughter cells from a single mother cell. The cell nucleus divides first and this is followed by the formation of a cell membrane between the daughter nuclei. In the type of cell division known as mitosis the daughter nuclei are identical to the original nucleus. Mitotic divisions cause growth but they ensure that all the cells of an individual are genetically identical to each other and to the original fertilized egg.

Any cell that through division eventually produces reproductive cells (the sperm or the ova, known as *gametes*) is known as a *germ* cell. In mammals the ovaries and the testes contain germ cells. Gametes are formed by germ cells by a cell division type called *meiosis*, which produces the reproductive cells each with half the number of chromosomes the mother cell had.

Meiosis may also cause the chromosomes of daughter cells to be different from that of the mother. The threadlike strand formed from a chromosome during the early stages of cell division is called a *chromatid*. Each chromosome divides along its length into two chromatids that subsequently separate completely. A process in which two chromatids "cross over" is an exchange of portions of chromatids between homologous chromosomes as chromosomes begin to move apart at the onset of meiosis. At certain contact points the two crossing chromatids break and rejoin in such a way that *sections* are exchanged. Crossing over thus alters the pattern of genes in the chromosomes. With gamete cells thus formed, the offspring would have a combination of characteristics different from that of the parents they came from.

Mutation is the other process essential for evolution. Mutation is a sudden random change in the genetic material of a cell that may cause it and all cells derived from it to differ in appearance or behavior from the normal cell. The organism affected by a mutation is called a *mutant*. Mutations occur naturally at a low rate but radiation and the action of some chemicals can increase it. Indeed the majority of

mutations harm the organism but a very small proportion may increase its fitness. Such improvements may spread subsequently over successive generations by natural selection.

During sexual reproduction, *recombination* (also called "crossover") of parental characteristics occurs naturally as follows. First, in *each* parent, genes are exchanged between each pair of chromosomes to form a single chromosome. Then, upon fertilization, the fusion of the nuclei of the male and female gametes occurs and the single chromosomes from each parent pair up to form a full set of diploid chromosomes in the offspring. In haploid sexual reproduction (the type emulated by most genetic algorithms), genes are exchanged between the two parents' *single-strand* chromosomes.

Since offspring are also subject to mutation, so not all of their traits are inherited. In mutation, single elementary bits of DNA are changed from parent to offspring, often attributed to "copying error." The fitness of an organism is typically defined as the probability that the organism will survive to reproduce, or as the number of offspring the organism produces.

In the simple genetic algorithm (the SGA) devised by Holland (1975), the term chromosome typically refers to a *coded* candidate solution to an optimization problem. SGA emulates primarily the process of natural selection after crossover and mutation. The solution is coded as a binary bit string or some other convenient representation. The "genes" are then either single bits, or short blocks of adjacent bits that encode a particular element (e.g., a variable x_i) in a multi-variable function optimization problem. An allele in a binary coded solution representation (a bit string) is the *value* of a gene—either 0 or 1. For larger alphabet problems (e.g., in scheduling problems) each gene can contain more than two alleles, such as all valid job numbers, machine identifications, or even production lot sizes. The site indicating the position of a gene held in a chromosome is called its *locus*.

As mentioned, many contemporary versions of GAs work with single-chromosome (haploid) "parents." Thus, crossover in the GA consists of exchanging genetic material between two haploid parents. Mutation is the flipping of a bit from 0 to 1 or vice versa, or replacing a symbol by a randomly chosen symbol at a randomly chosen locus.

The genotype of an individual solution in GA using binary bit strings is simply the configuration of bits in that individual's chromosome. There is usually no separate notion of phenotype in the context of GAs; often it is the same. For instance, the genotype of a job-sequence chromosome in flowshop scheduling can be 1 7 3 5 6 2 4 whereas its phenotype will be the plan "job 1 is processed first, then job 7, then job 3, etc..."

Holland's GA design assumes that high-fitness parent solutions from different regions in the search space can be combined by crossover to produce, on occasion, high-fitness offspring candidate solutions. The *fitness landscape* is a representation of all possible genotypes along with their fitness. In GAs crossover and mutation are ways of moving a population around on the fitness landscape. And, even though the details may differ a little, most evolutionary computation methods called "GAs" use a population of chromosomes (solutions), selection according to fitness, crossover to produce new offspring, and random mutation of the new offspring. Note also that the majority of contemporary GA implementations use variations of selection, crossover and mutation. A chromosome mutation caused by the reversal of part of a chromosome, so that the genes within that part are in inverse order, is *inversion*. Inversion mutations usually occur during crossing over in meiosis. Inversion is rarely used in GA. Additional processes now incorporated are *niche formation, sharing* and *speciation*. These are powerful processes seen to be active in nature and that lead to the formation of stable populations of multiple species.

Species branch off from ancestral stocks, usually as small, discrete populations inhabiting a definite geographical area. They establish their uniqueness by evolving a genetic program sufficiently distinct that members of the species breed with each other, but not with members of other species. Their members share a common ecological niche and continue to interact through interbreeding (Gould, 1993). New species usually arise not by the slow, steady transformation of the entire ancestral populations, but by the splitting off of small isolates from an unaltered parental stock (Weiner, 1994). Gould provides some fascinating related theories and their interpretations. We discuss these processes in Chapters 7 and 8 of this text.

In summary, GAs as used today are computerized search and optimization algorithms inspired by the mechanics of natural genetics

and natural selection. Thus, GAs are fundamentally different from classical optimization algorithms in several respects. In the following pages we provide a sketch of the working principles of the SGA and also a computer simulation given by Deb (1995) that illustrates these principles.

2.2 WORKING PRINCIPLES

Consider the following unconstrained optimization problem:

$$\text{Maximize } f(\mathbf{x}), \quad x_i(L) \leq x_i \leq x_i(U), \quad i = 1, 2, 3, \dots, N.$$

Here \mathbf{x} is an N -dimensional vector of decision variables x_1, x_2, \dots, x_N . Even though we address a maximization problem here, GAs can handle minimization problems equally deftly. To solve the above problem by SGA, the first normal step is the coding of the decision variables $\{x_i\}$ in some suitable "string" structure akin to biological chromosomes. Note, however, that this coding is not absolutely necessary even though coding provides control over the precision of the solution being found. It is often possible to run GAs using the original real-valued (or categorical) variables directly (Deb, 1995).

Binary-coded strings or representations utilizing 1's and 0's are frequently used in GA applications to numerical optimization. The length of the string is usually set to meet the required precision (hence the likelihood of accuracy) of the final solution. For example, if four bits are used to represent each x_i in a two-variable optimization problem, then the strings 0000 0000 and 1111 1111 would represent the points

$$(x_1(L), x_2(L))^T \text{ and } (x_1(U), x_2(U))^T$$

respectively because the substrings 0000 and 1111 have the minimum and maximum decoded values in the binary representation of decimals. Any other 8-bit string between 0000 and 1111 can now represent a point in the (x_1, x_2) search space. Subsequently, if the real decision variable x_i were represented by the binary substring s_i of length l_i then the "mapping rule" to return from the binary representation to real variables would be

$$x_i = x_i(L) + [x_i(U) - x_i(L)] / [2^{l_i} - 1] \text{ decoded value of } s_i \quad (2.1)$$

The complete solution string s is represented as $(s_{l-1} s_{l-2} s_{l-3} \dots s_2 s_1 s_0)$. The decoded value of substring s_i is calculated here as

$$\sum_{i=0}^{l-1} 2^i s_i$$

where $s_i \in \{0, 1\}$.

For example, a 4-bit string 0111 (i.e., $l = 4$) has the decoded value equal to

$$1 \times 2^0 + 1 \times 2^1 + 1 \times 2^2 + 1 \times 2^3 = 7.$$

Note that with four bits used in the above coding scheme, only 2^4 or 16 distinct substrings are possible. This divides the search space into 16 equally spaced parts. If the string length is increased by one (i.e., l is made 5), the precision of search improves to $1/32^{\text{th}}$ of the search space between $x_i(U)$ and $x_i(L)$. Generalizing this scheme we see that with an l_i -bit coding scheme, the obtainable precision of search is

$$[x_i(U) - x_i(L)]/2^{l_i}$$

However, it is not necessary to code all decision variables $\{x_i\}$ into equal-sized substrings. Once a coding scheme has been decided upon, the real values of the decision variables $x_1, x_2, x_3 \dots x_N$ may be found using the mapping rule (2.1). The function value $f(x)$ corresponding to the "point" x may be then calculated by substituting $x_1, x_2, x_3 \dots$ in $f(x)$.

2.2.1 Fitness Function

As mentioned above, GAs emulate or mimic the survival-of-the-fittest principle of nature to accomplish the search process. Maximization is therefore straightforward for GAs. Minimization problems, however, require some transforming.

To achieve maximization, a *fitness function* $F(x)$ is first derived from the problem's objective function $f(x)$ and one sets $F(x) = f(x)$. For minimization, several different transformations are possible. One popular transformation is

$$\mathcal{F}(x) = 1 / (1 + f(x))$$

Note that this transformation does not alter the location of the minimum, but it converts the original minimization problem into a maximization problem. The fitness function value $\mathcal{F}(x)$ is known as the string x 's *fitness*.

2.3 THE GENETIC SEARCH PROCESS

GAs begin with a population of randomly selected initial solutions—a population of random strings representing the problem's decision variables $\{x_i\}$. Thereafter, each of these initially picked strings is evaluated to find its fitness. If a satisfactory solution (based on some acceptability or search stoppage criterion) is already at hand, the search is stopped. If not, the initial population is subjected to genetic evolution—to procreate the next generation of candidate solutions (Spillman, 1993).

The genetic process of procreation uses the initial population as the input. The members of the population are "processed" by four main GA operators—reproduction, crossover, mutation and inversion—to create the progenies (the next generation of candidate solutions to the optimization problem at hand). The progenies are then evaluated and tested for termination. If the termination criterion is not met, the three GA operators iteratively operate upon the population. The procedure is continued till the termination criterion is met. One cycle (iteration) of these operations to produce offspring is called a *generation* in the GA terminology. The different GA operators function as follows.

2.3.1 Reproduction

This is usually the first operator that is applied to an existing population to create progenies. Reproduction first *selects* good parent solutions or strings to form the *mating pool*. The essential idea in reproduction is to select strings of above-average fitness from the existing population and insert their *multiple* copies in the mating pool, in a probabilistic manner. This results in a selection of existing

solutions with better-than-average fitness to act as parents for the next generation.

One popular reproduction operator uses fitness proportionate selection in which a parent solution is selected to move to the mating pool with a probability that is proportional to its own fitness. Thus, the i^{th} parent would be selected randomly with a probability p_i proportional to \mathcal{F}_i , given by

$$p_i = \mathcal{F}_i / \sum \mathcal{F}_j$$

This scheme is also known as the *roulette-wheel* selection scheme and it mimics the survival-of-the-fittest rule. Treated in this manner, a solution that has a high fitness will have a higher probability of being copied into the mating pool and thus participate in parenting offspring. On the other hand, a solution with a smaller fitness value will have a smaller probability of being copied into the mating pool.

Notice that reproduction is merely copying certain parents to form the mating pool. It *does not* create new solutions! In search, however, the creation and evaluation of new solutions is a primary mission. To achieve this the GA uses three other operators to modify the solution strings at hand and thus produce new strings in the mating pool, as follows.

2.3.2 Crossover

Crossover is also known as "recombination." In the crossover operation exchanging information among the strings present in the mating pool creates new strings (solutions). Recall that a string representation of a solution contains "information" in its "genes"—the bits that make up the body of the string. For instance, the string 010111 contains specific information in the six distinct positions, as do chromosomes in natural genetics. In crossover, two strings are picked from the mating pool and some portions of these strings are exchanged between them. A common implementation of "crossover" uses the single-point crossover process in which a crossing site is randomly chosen along the string length and all bits to the right side of this crossing site are exchanged between the two parent strings as shown below.



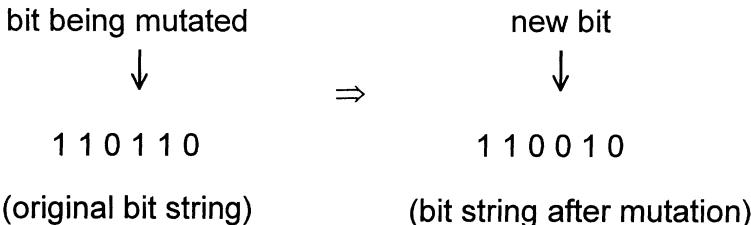
It is expected from the above operation that if good substrings from the parents get combined by crossover, the children are likely to have improved fitness. Further, because a "pressure" is exerted by survival-of-the-fittest selection method used in forming the mating pool, when good strings are created in the progenies by crossover, they help the search process by propagating their superior characteristics in the ensuing generations. However, since the location of the appropriate (i.e. best) site of crossover cannot be known *a priori* without considerable additional computation, a random site is chosen in practice.

Still, the effect of crossover can be detrimental *or* beneficial. Therefore, to preserve some of the good strings in the mating pool, not all strings in the mating pool are used in crossover. A crossover probability (p_c) is used to decide whether a given member of the mating pool will be crossed. And, on an average $100(1-p_c)$ per cent of the mating pool pass on as they are to the next generation.

2.3.3 Mutation

While a crossover operator attempts to produce new strings of superior fitness by effecting large changes in a string's makeup (this is akin to large jumps in search of the optimum in the solution space), the need for local search around a current solution also exists. This is accomplished by *mutation*. Mutation is additionally aimed to maintain diversity in the population. (We shall see later in this text that resource or fitness sharing is another mechanism for maintaining diversity in a population.)

Mutation creates a new solution in the neighborhood of a current solution by introducing a small change in some aspect of the current solution. In practice, for example, if binary coding is being used, a single bit in a string may be altered (from 0 to 1, or 1 to 0) with a small probability, creating a new solution. Thus,



Crossover aims at recombining parts of good substrings from good parent strings to hopefully create a better offspring. Mutation, on the other hand, alters a single child string locally to hopefully create a superior child string.

2.3.4 Inversion

Inversion was a GA operator originally suggested by Holland (1975) to also produce new strings from a few existing strings—by making large changes. Inversion in nature reverses the order of a contiguous section of the chromosome, thus rearranging the order in which those genes were originally placed. However, the use of inversion as a search device has not been as popular as crossover.

Even though the claims for producing superior strings by crossover, mutation or inversion are no guarantees, still, it is expected that superior strings when created will survive and propagate their characteristics, while the inferior new strings produced will be eliminated quickly. This expectation is embodied in a mathematical assertion called Holland's "Schema Theorem" (see Section 2.7 below).

The genetic algorithm has now been constructed in a variety of forms. We present the "simple" genetic algorithm in a step-by-step format, as follows.

2.4 THE SIMPLE GENETIC ALGORITHM (SGA)

- Step 1** Select a coding scheme to represent the optimization problem's decision variables. Devise also an appropriate selection operator, a crossover operator, and a mutation operator to work with the selected coding scheme. Choose population size p_s , crossover probability p_c , and mutation probability p_m . Initialize a random population of strings. Choose a maximum allowable generation number t_{\max} . Set $t = 0$.
- Step 2** Evaluate the fitness (\mathcal{F}) of each string in the population.
- Step 3** If $t > t_{\max}$ or some other search termination criterion is satisfied, terminate.
- Step 4** Perform reproduction to create the mating pool.
- Step 5** Perform crossover on random pairs of strings taken from the mating pool.
- Step 6** Perform mutation on every string.
- Step 7** Evaluate strings in the new population. Set $t = t + 1$ and go to Step 3.

GAs may be used to optimize numerical functions of continuous or discrete variables as well as discrete fitness measurements such as those arising in combinatorial optimization. Applications of GA now occur in engineering, economics, production management, applied mathematics, and several other fields involving a large number of potential solutions. In the following pages we illustrate the working of the genetic algorithm using the optimization of an unconstrained Himmelblau function (Reklaitis, Ravindran and Ragsdell, 1983) as the test problem.

2.5 AN APPLICATION OF GA IN NUMERICAL OPTIMIZATION

This example is obtained from Deb (1995). The objective here is to minimize the two-variable Himmelblau function

$$f(x_1, x_2) = (x_1^2 + x_2 - 11)^2 + (x_1 + x_2^2 - 7)^2 \quad (2.2)$$

in the interval $0 \leq x_1, x_2 \leq 6$. For reference, the true solution to this problem is $(3, 2)^T$ having a function value equal to zero (Figure 2.1).

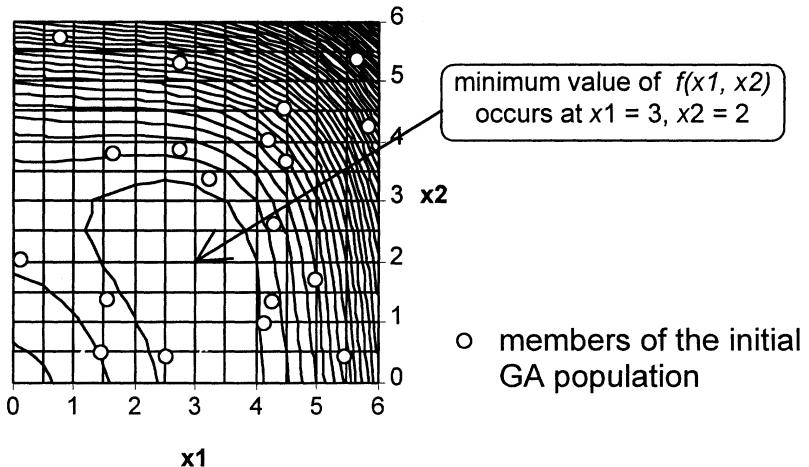


FIGURE 2.1 THE INITIAL POPULATION AND THE CONTOURS OF $f(x_1, x_2)$

Step 1 We decide to use binary coding for the variables x_1 and x_2 . We set a precision target of ± 0.006 for both x_1 and x_2 , in the interval $[0, 6]$. This gives us binary substrings of length 10 for both x_1 and x_2 , resulting in a total string length (l) of 20 bits. Further, we select roulette-wheel selection, a single-point crossover operator, and a bit-wise mutation operator. The details of these operations are shown below. The crossover probability p_c is set at 0.8 while the mutation probability p_m is set at 0.05. This implies that any two strings chosen randomly from the mating pool will have a 0.8 probability of crossing and producing progeny. Also, the chance of any given bit in a string mutating (i.e., a 0 becoming 1 or vice versa) is 0.05. (Note that this arrangement allows *more than* one bit on a given string to be mutated.) Population size p_s is set at 20 while the algorithm terminates at $t_{\max} = 30$.

TABLE 2.1 THE INITIAL RANDOMLY GENERATED POPULATION

String #	Substring 2	Substring 1	x_2	x_1	$f(x_1, x_2)$	$\mathcal{F}(x_1, x_2)$	Expected Count	Selection Probability	Cumulative Prob of Selection
1	1100100000	1100100000	5.349	4.692	959.680	0.001	0.13	0.007	0.007
2	0001001101	0011100111	0.452	1.355	105.520	.009	1.10	0.055	0.062
3	1010100001	0111001000	3.947	2.647	126.685	0.008	0.98	0.049	0.111
4	1001000110	1000010100	3.413	3.120	65.026	0.015	1.85	0.093	0.204
5	1100011000	1011100011	4.645	4.334	512.197	0.002	0.25	0.013	0.217
6	0011100101	0011111000	1.343	1.455	70.849	.014	1.71	0.086	0.303
7	0101011011	0000000111	2.035	0.041	88.273	.011	1.34	0.067	0.370
8	1110101000	1110101011	5.490	5.507	1436.563	.001	0.12	0.006	0.376
9	1001111101	1011100111	3.736	4.358	265.556	.004	0.49	0.025	0.401
10	0010100100	1010101010	0.962	4.000	39.849	.024	2.96	0.148	0.549
11	1111101001	0001110100	5.871	0.680	814.117	.001	0.14	0.007	0.556
12	0000111101	0110011101	0.358	2.422	42.598	.023	2.84	0.142	0.698
13	0000111110	1110001101	0.364	5.331	318.746	.003	0.36	0.018	0.716
14	1110011011	0110000010	5.413	2.639	624.164	.002	0.24	0.012	0.728
15	1010111010	1010111000	4.094	4.082	286.800	.003	0.37	0.019	0.747
16	0100011111	1100111000	1.683	4.833	197.556	.005	0.61	0.030	0.777
17	0111000010	1011000110	2.639	4.164	97.699	.010	1.22	0.060	0.837
18	1010010100	0100001001	3.871	1.554	113.201	.009	1.09	0.054	0.891
19	0011100010	1011000011	1.326	4.147	57.753	.017	2.08	0.103	0.994
20	1011100011	1111010000	4.334	5.724	987.955	.001	0.13	0.006	1.000

The initial population is created by Knuth's (1983) random number generator, with a seed set at 0.760. Figure 2.1 and Table 2.1 display this population. Note the location of the global optimum ($x_1 = 3.0$, $x_2 = 2.0$) in the (x_1, x_2) plane. Note also the locations of the randomly generated initial population. As GA progresses, we shall track the evolution of new solutions—the GA descendants of these initial solutions.

Step 2 The next step is to evaluate each string in the population. The first string has two substrings (1100100000 and 1110010000) which decode into $2^9 + 2^8 + 2^5$ and $2^9 + 2^8 + 2^7 + 2^4$ (= 800 and 912) respectively. These decoded values must in turn be converted into real values for x_1 and x_2 . Thus, for x_1 we have the value $0 + (6 - 0)800/1023$ or 4.692 and for x_2 we have $0 + (6 - 0)912/1023$ or 5.349. These values when substituted into (2) give $f(x_1, x_2) = 959.680$. The fitness function \mathcal{F} for the solution ($x_1 = 4.692$ and $x_2 = 5.349$) may be now calculated. This equals $1.0/(1.0 + 959.680)$, or 0.001. This fitness function value determines the degree to which the first string will participate in parenting the subsequent generation. The fitness of the other nineteen strings in the initial population are calculated in the similar manner. Column $\mathcal{F}(x_1, x_2)$ of Table 2.1 shows these values.

Step 3 Since $t = 0$ is less than t_{\max} , we proceed to Step 4.

Step 4 This is the Selection/Reproduction step of the GA in which the mating pool is created by selecting the higher fitness strings in the population. Several selection methods may be used here. We illustrate presently the roulette wheel selection procedure. In this procedure first the average fitness F_{avg} of the population is calculated. Here it is 0.008. The next step determines the expected count in the mating pool to be created of each string present in the current population. This count equals F/F_{avg} , indicating that the higher is the fitness of a given string, the higher is the expected number of copies it is to have in the mating pool. Column "Expected Count" in Table 2.1 shows these count values for the different strings in the population.

The roulette-wheel probability of a particular string's being copied to the mating pool is calculated by dividing the count value for that string by the population size. This is the probability of the string's getting selected. Column "Selection Probability" in Table 2.1 shows these probabilities.

The roulette wheel selection method to create the mating pool works as follows. First the cumulative probability of selection (shown in Column "Cumulative Prob of Selection") for the different strings is calculated. Next, to determine whether a particular string would actually get selected, a random number between 0 and 1 is drawn and then compared to the cumulative probabilities of the different strings, as follows.

Suppose that the random number drawn is 0.472. This value (shown in Column 1 in Table 2.2) falls between 0.401 (the cumulative selection probability value of string #9) and 0.549 (the cumulative probability value of string #10). This would lead to a copy of string #10 being selected for its inclusion in the mating pool. Since a total of twenty members would form the mating pool, this random selection method is repeated twenty times. The result of such twenty repeated random selection trials is shown in Table 2.2.

Column 2 of Table 2.2 indicates the "number" of the string tagged for reproduction in each of the twenty random trials. Columns 3 and 4 together show the "mating pool," the constituents of the actual strings that got selected and then reproduced. The mating pool thus formed would parent the members of the subsequent generation through crossover and mutation operations. Figure 2.2 shows the initial population and also the mating pool members.

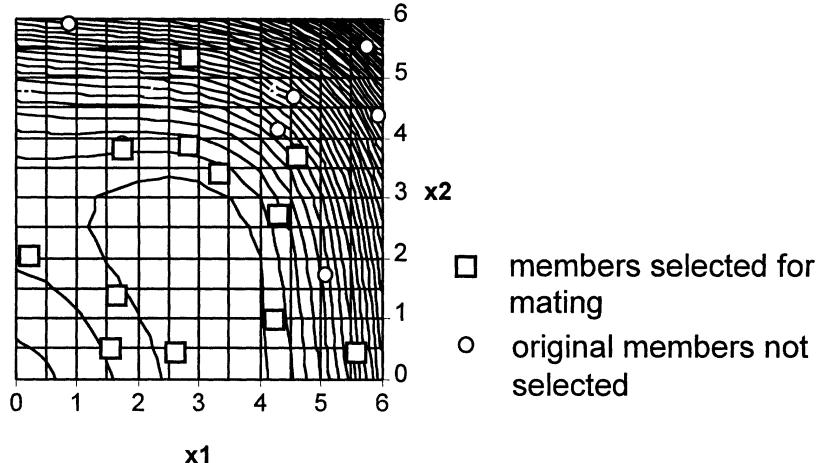


FIGURE 2.2 MEMBERS SELECTED TO FORM THE MATING POOL

TABLE 2.2 THE MATING POOL FORMED BY REPRODUCTION

Random Number Drawn	Selected String's String# in Original Pool	Reproduced Mating Member's Substring 2	Reproduced Mating Member's Substring 1
0.472	10	0010100100	1010101010
0.108	3	1010100001	0111001000
0.045	2	0001001101	0011100111
0.723	14	1110011011	0111000010
0.536	10	0010100100	1010101010
0.931	19	0011100010	1011000011
0.972	19	0011100010	1011000011
0.817	17	0111000010	1011000110
0.363	7	0101011011	0000000111
0.189	4	1001000110	1000010100
0.222	6	0011100101	0011111000
0.288	6	0011100101	0011111000
0.615	12	0000111101	0110011101
0.712	13	0000111110	1110001101
0.607	12	0000111101	0110011101
0.192	4	1001000110	1000010100
0.386	9	1001111101	1011100111
0.872	18	1010010100	0100001001
0.589	12	0000111101	0110011101
0.413	10	0010100100	1010101010

You may notice that String #14 ($F = 0.002$) got selected here but String #16 ($F = 0.005$) did not, even though String #16 had a higher fitness. This is why the roulette wheel selection procedure is called "noisy." More stable versions of selection are available, and will be described later in this chapter.

Step 5 This is the crossover step. Here the members of the mating pool are caused to mate or probabilistically exchange portions of their "genes" (as coded in their substrings) to produce progenies with characteristics inherited from their parents. Since random picking created the mating pool, we decide to select pairs of strings for crossover from the top of Table 2.2. Thus, strings #3 and #10 participate in the first crossover. First, a "coin with probability of producing head = 0.8" is flipped to see if the picked pair would mate. If the result is "yes," the pair is crossed; otherwise it is placed as is in an *intermediate population* for subsequent genetic processing. If a pair is to be crossed, a random cross-site is chosen by drawing a random number between 0 and $(l - 1)$, which here is between 0 and 19. It turns out that this random number is 11, hence we cross string #3 and string #10 at site 11.

After crossing, the resulting progenies are placed in the intermediate population. The results of crossover are shown in Table 2.3.

In Figure 2.3 the results of processing the complete mating pool by crossover are displayed. Notice that while some crossovers created points away from the optimum solution, other crossovers produced solutions that are closer.

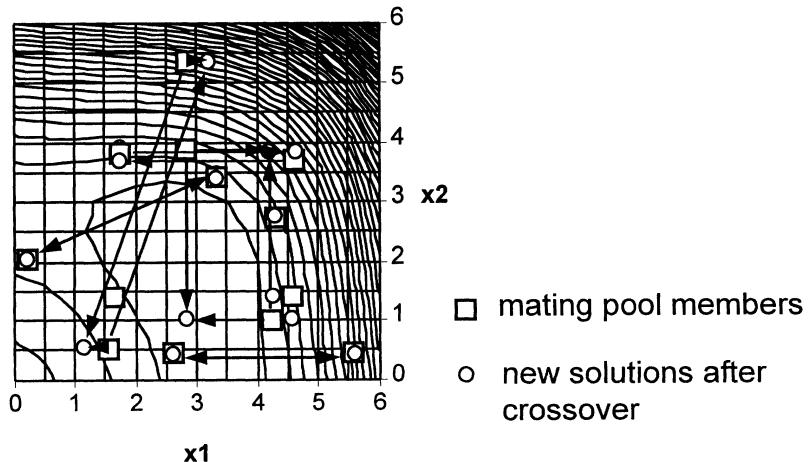


FIGURE 2.3 THE POPULATION AFTER CROSSOVER

TABLE 2.3 INTERMEDIATE POPULATION FORMED BY 14 CROSSEOVERS

Mating Member's Substring 2	Mating Member's Substring 1	Crossover to occur?	Crossing site	Intermediate Member's Substring 2	Intermediate Member's Substring 1
0010100100	1010101010	Yes	9	0010100101	0111001000
1010100001	0111001000	Yes	9	1010100000	1010101010
0001001101	0011001111	Yes	12	0001001101	0011000010
1110011011	0111000010	Yes	12	1110011011	0111000111
0010100100	1010101010	Yes	5	0010100010	1011000011
0011100010	1011000011	Yes	5	00111000100	10101001010
0011100010	1011000011	No		0011100010	1011000011
0111000010	10110000110	No		0111000010	1011000110
0101011011	00000000111	Yes	14	0101011011	00000010100
1001000110	00000010100	Yes	14	1001000110	1000000111
0011100101	0011111000	Yes	1	0011100101	0011111000
0011100101	0011111000	Yes	1	0011100101	0011111000
0000111101	0110011101	No		0000111101	0110011101
0000111110	1110001101	No		0000111110	1110001101
0000111101	0110011101	Yes	18	0000111101	0110011100
1001000110	10000010100	Yes	18	1001000110	10000010101
1001111101	1011100111	Yes	10	1001111101	0100001001
1010010100	0100001001	Yes	10	1010010100	1011100111
0000111101	0110011101	No		0000111101	0110011101
0010100100	1010101010	No		0010100100	1010101010

Step 6 The next step performs mutation on strings in the intermediate population. For bit-wise mutation of each string, a coin with the probability of producing head ("mutation will occur") equal to 0.05 is flipped for *every* bit. If the outcome of this flip is "head," the bit is mutated (from 0 to 1 or from 1 to 0). Since there are twenty members being processed, each having 10+10 bits, we expect about $0.05 \times 20 \times 20$ bits to alter.

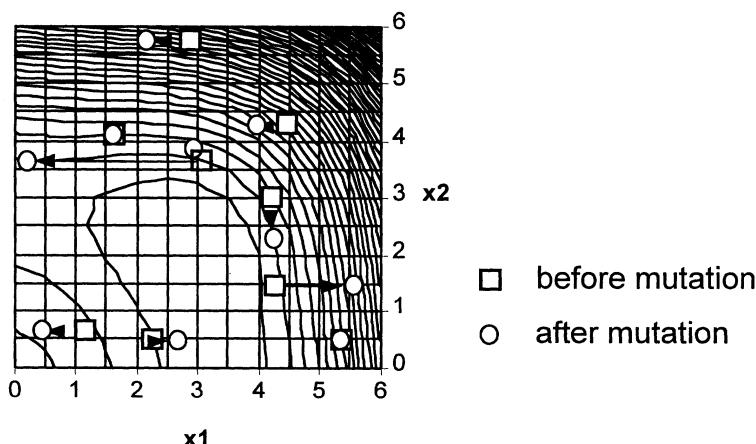


FIGURE 2.4 THE POPULATION AFTER A FEW MUTATIONS

Columns 1 and 2 of Table 2.4 show the substring bits (in *italic*) that actually mutated, the count being 16. Figure 2.4 shows the state of the population after mutation.

Again, it should be noted that mutation created some better and some worse solutions. Indeed, production of some worse solutions by mutation is intentional. It maintains diversity in the population and reduces the probability of premature convergence of the GA to a local optimum.

TABLE 2.4 POPULATION FORMED AFTER MUTATION WITH MUTATED BITS SHOWN IN ITALICS

Member's Substring 2 After Mutation	Member's Substring 1 After Mutation	x_1	x_2	$f(x_1, x_2)$	Fitness $f(x_1, x_2)$
001010101	0111001000	1.015	2.674	18.886	0.050
1010100001	1010101010	3.947	4.000	238.322	0.004
0001001101	0001000010	0.452	0.387	149.204	0.007
1110011011	0101100011	5.413	2.082	596.340	0.002
0010100010	1011000011	0.950	4.147	54.851	0.018
0011100100	1110101010	1.337	5.501	424.583	0.002
0011100011	1011100011	1.331	4.334	83.929	0.012
0101010010	1011000110	1.982	4.164	70.472	0.014
0101011011	0000010100	2.035	0.117	87.633	0.011
1001010110	1000000111	3.507	3.044	72.789	0.014
0011100101	0011111000	1.343	1.455	70.868	0.014
0011100101	0011111000	1.343	1.455	70.868	0.014
0000101101	0111011100	0.264	2.792	25.783	0.037
0000111110	1110001101	0.364	5.331	318.746	0.003
0000111101	0110001100	0.358	2.416	42.922	0.023
1001000110	0000010101	3.413	0.123	80.127	0.012
1001111101	0100001001	3.736	1.554	95.968	0.010
1010010100	1010100111	3.871	3.982	219.426	0.005
0000111101	0110011101	0.358	2.422	42.598	0.023
0010100100	1010101010	0.962	4.000	39.849	0.024

Step 7 The population resulting from selection, reproduction, crossover and mutation becomes the *new* population. To complete the computational cycle, we evaluate each string (as done in Step 2) by first identifying the substrings in each variable and then decoding them. The generation count is incremented to $t = 1$ and the algorithm proceeds to Step 3 to produce the next generation.

The average fitness F_{avg} of the initial population was 0.008. The average fitness of the population after crossover and mutation is 0.015, a considerable improvement. The best solution ($x_1 = 4.000$, $x_2 = 0.962$) in the initial population had a fitness of 0.024. The best solution ($x_1 = 2.674$, $x_2 = 1.015$) in the new population has fitness 0.050, also a significant improvement. The population after 25 generations is shown in Figure 2.5. At this point the best solution is found to be $x_1 =$

$x_1 = 3.003$ and $x_2 = 1.994$, giving $f(x_1, x_2) = 0.001$ and fitness $F = 0.999$. Figure 2.5 shows the clustering of the population (near the optimum solution $x_1 = 3.000$, $x_2 = 2.000$) that resulted when the GA terminated. The average total function evaluations required to this point is $0.8 \times 20 \times 26 = 416$.

A problem solving methodology would be *robust* if it produces the same answer every time the procedure is repeated. The robustness of the GA in seeking out the global optimum may be demonstrated by starting the algorithm using different starting solutions in Step 1. We note that this robustness is greatly affected by the choice of the parameters p_s , p_c , p_m and t_{\max} (a topic addressed separately in Chapter 3 of this text).

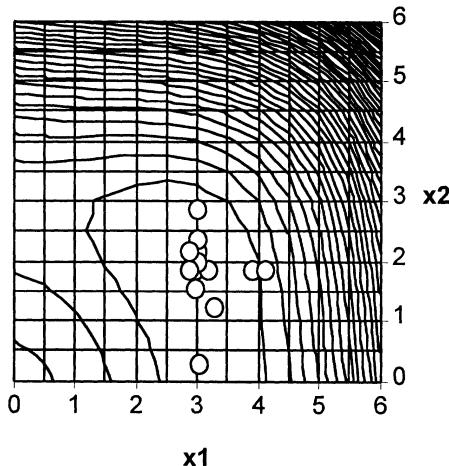


FIGURE 2.5 THE POPULATION CLUSTERED NEAR THE OPTIMUM AFTER 25 GENERATIONS

2.6 GENETIC ALGORITHMS vs. TRADITIONAL OPTIMIZATION

Like GAs, some traditional methods such as Box's EVOP (evolutionary optimization, see Box, 1957) method are population-based. But, these methods do not use previously obtained information formally. In GAs, previously found good information is emphasized during proportional reproduction and it is propagated

through crossover and mutation operators. Also, GAs permit capturing of multiple optimal solutions. Such solutions can facilitate multiobjective function optimization.

GA operators, unlike gradient-based and other similar methods, do not work deterministically. Almost without exception, traditional methods use fixed transition rules to move from one solution to another. By contrast, reproduction in GA uses roulette wheel selection while crossover and mutation both use probabilistic transitions. Thus, the traditional methods may be highly dependent on the starting solution while GAs, when appropriately parameterized, can exhibit a great deal of robustness.

In fact, a properly parameterized GA run does not get trapped in a local optimum point. It also converges to the global optimum with a high degree of consistency, regardless of the specification of the initial population. Since the crossover points are chosen at random, many search directions are possible when the GA executes. Some of these directions may lead to the global basin while others may not. However, the reproduction operation indirectly filters good directions and helps guide the search. The search in the mutation operator is similar to a local search around a current good solution such as the local exploratory search done in the Hooke-Jeeves method.

However, traditional optimization methods can be very effective in solving special classes of problems. GAs can be rather slow in these applications. For instance, gradient search methods can outperform almost any algorithm in solving continuous, unimodal problems even though such methods are not effective with multi-modal problems.

About the time when Darwin put forth his "theory" of evolution based on the principle of natural selection and inheritance, several alternative explanations of evolution were proposed. One prominent theory is due to Jean-Baptiste de Lamarck. Lamarck put forth an alternative to Darwin's natural selection theory and proposed that environmental changes throughout an organism's life cause structural changes that are transmitted to the offspring. Lamarck's theory of evolution (known as Lamarckism) lets organisms pass along the knowledge and experience they acquire in their lifetime to their offspring. However, no biologist today believes that traits thus acquired during one's lifetime (such as strong right arm of a ironsmith) can be genetically transmitted to the offspring. The

"Central Dogma" (see Sections 4.8 and 7.1) opposes such possibility. Still, if we look at *cultural evolution* (the generation of ideas and knowledge by humankind, for instance), we do see that certain qualities get passed on to offspring through learning, teaching, etc., albeit nongenetically.

Indeed structural language and culture allow humans to pass on the benefits acquired through experience and learning, one generation to the next, even though such transfer is not accepted today to be genetic. It is possible to incorporate such "learning" into the *artificial* world of genetic algorithms. Such improvements acquired during the "lifetime" of a generation can actually be incorporated easily into the makeup of the artificial organisms with which GA works.

Such a method is clearly not a Darwinian GA. Rather, it is hybrid. In this method the artificial organisms would first pass through biological evolution and then pass through a Lamarckian cultural or intelligent evolution in each generation.

One such hybrid scheme would operate as follows. To keep the scheme simple, we use here only the recombination (crossover) operator. In the scheme shown, $P(t)$ is the parent population of the beginning of first generation t . A progeny population ($C^*(t)$) is produced by genetic recombination of suitably chosen parents in $P(t)$. Then $C^*(t)$ is improved, whenever possible, using an operation such as hill climbing or EVOP, applied on each or a few randomly members of $C^*(t)$, to evolve progeny $C(t)$ with improved fitness. Subsequently, the common pool consisting of $P(t)$ and $C(t)$ is subjected to selection to yield the parents of the next generation, $P(t+1)$. The scheme is shown on the next page. Many other variations are clearly possible.

Several variants of the above scheme have been proposed recently in which attempt has been made to inject some "smarts" (problem-domain specific improvements) in the initial offspring $C^*(t)$ before accepting it. Many researchers report superior on-line as well as off-line performance of hybridized GAs. Sections 4.8 and 4.9 in Chapter 4 discuss an effective scheme for hybridizing GAs for flowshop scheduling.

```

Procedure Hybrid GA
begin
    t ← 0
    initialize P(t)
    evaluate P(t)
    while (not termination condition) do
        recombine P(t) to yield C*(t)
        evaluate C*(t)
        locally climb C*(t) to yield C(t) using problem-specific
        knowledge
        select P(t+1) from P(t) + C(t)
        t ← t + 1
    end
end

```

2.7 THEORETICAL FOUNDATION OF GAs

It is natural to ask how GAs work. Before we probe this question, however, we should recognize that "evolution" as being contrived by GAs is still perhaps a very early attempt at constructing a meta-heuristic to adaptively seek optimum solutions. Important issues in explaining how GAs work still remain open and controversial. The original GA using binary strings was invented and an explanation of how it works was given by Holland (1975) in the form of his famous "schema theorem." Later Goldberg (1989), Rawlings (1991), Whitley and Vose (1995), Mitchell (1996) and Muhlenbein (1997) added a great deal to this original explanation. Tackett (1994) attempted extensions to this explanation for non-binary strings. Vose (1993), on the other hand, has developed a Markov chain model for elementary GAs. Muhlenbein (1997) uses population genetics. In the following paragraphs we abstract Mitchell's lucid narration of how GAs work, including the limitations that keep this from being the "last word" on GAs.

2.7.1 Building Blocks

GAs are simple to describe and program, yet the ensuing search process is a complicated stochastic process. It is also not yet clear as to what kind of problems GAs are most suited for, what controls their

convergence rate, and what precisely are the roles of crossover, mutation, inversion, etc., in the overall search in progress. Holland's original theory assumed that GAs work by discovering, emphasizing, and recombining good "building blocks" (combination of bit values that confer higher fitness on strings containing them) of solutions in a highly parallel manner. Holland's GA focuses on suitably "processing" these building blocks using the operators of selection, one-point crossover and mutation.

Holland formalized the informal notion of building blocks into *schemas*. Holland's schema is a set of bit strings best described as templates made up of binary bits (0 and 1) and asterisks (*). An example is $1^{***}1$, a 6-bit schema in which "*" in positions 2 to 5 imply the use of "wild cards" or "we don't care what value they possess." Goldberg (1989) in his text on GAs used the notion of hyperplanes for schemas. A hyperplane H is a "plane" of various dimensions (dependent on the length of the schema) in the l -dimensional search space of l -bit strings.

Note that not every possible subset of the set of length l bit strings can be called a schema. Thus in the binary coding scheme, there are only 3^l possible schemas. A GA population of n strings contains between 2^l and $n \times 2^l$ different schemas.

2.7.2 Order and Defining Length

The schema $1^{***}1$ has two *defined* (fixed) bits—the first and the 6th bit. This attribute (the number of defined bits) is called the *order* of the schema. Thus, the order of H (written as $o(H)$) is 2. The other important attribute of a schema, according to Holland, is its *defining length*, the distance between its outermost defined bits. The defining length of H (written as $d(H)$) is 5. The different strings that fit a schema are known as its *instances*. Thus, 100111 and 110011 are both instances of H .

With binary coded strings, each position or site in a schema can have three possible values (0, 1 or *). Thus, 3^l distinct schemas are possible for bit strings of length l . On the other hand, any given bit string of length l (with all positions defined) is an instance of only 2^l different schemas. Thus, the string 11 is an instance of schemas **, *1, 1* or 11.

In this way it may be seen that any given set (collection or population) of n defined strings contains instances of between 2^l (all strings here are alike) and $n \times 2^l$ (each string here is distinct) different schemas.

The last assertion has a rather important implication. While the GA is evaluating the fitness of only n strings explicitly, it is actually *implicitly* estimating the average fitness of a much larger number of schemas. The average fitness of a schema here is defined as the average fitness of all possible instances of that schema. Thus, if n strings are generated randomly, we would expect about half of them to be instances of $1^{***}...*$ and the other half to be instances of $0^{***}...*$. The average fitness of the instances of $1^{***}...*$ present in the population would then give an estimate of the average fitness of this schema (because the sample would probably not contain all possible instances of $1^{***}...*$). An important point made here by Holland is that just as the GA does not explicitly represent schemas, the estimates of schema average fitness are neither calculated nor stored by the GA. Still, the GA *behaves* as if it were actually calculating and storing these averages.

Owing to the survival-of-the-fittest selection scheme that the GA actuates in each generation, it encourages the growth in the instances of the higher fitness schemas and discourages such growth for low fitness schemas. Holland modeled this growth dynamics and he derived a fundamental condition for the growth of high fitness schemas, as follows.

2.7.3 The Schema Theorem

A lower bound on the growth of good ("higher-than-population-average fitness") schemas may be derived by assuming the presence of H , a schema with at least one instance present in the population at time t . Let $m(H, t)$ be the number of instances of H at time t and $u(H, t)$ be the observed average fitness of H (calculated using all the instances of H present in the population at time t) at time t . The object is to calculate $E[m(H, t + 1)]$, the expected number of instances of H at time $t + 1$. Selection is assumed here to be fitness-proportionate: the expected number of offspring of a string x is equal to $\mathcal{F}(x)/\mathcal{F}_{avg}(t)$, where $\mathcal{F}(x)$ is the fitness of x and $\mathcal{F}_{avg}(t)$ is the average fitness of *all* strings present in the population at time t .

Recall now that

$$\mathcal{F}_{\text{avg}}(t) = \sum_{\text{all } x} \mathcal{F}(x) / n$$

Hence

$$\begin{aligned} E[m(H, t+1)] &= \sum_{x \in H} \mathcal{F}(x) / \mathcal{F}_{\text{avg}}(t) \\ &= m(H, t) u(H, t) / \mathcal{F}_{\text{avg}}(t) \end{aligned}$$

because $u(H, t) = \sum_{x \in H} \mathcal{F}(x) / m(H, t)$ by definition of $u(H, t)$.

Thus, even though the GA does not calculate $u(H, t)$ explicitly, the increase or decrease in the average number of schema instances of H (i.e. $E[m(H, t+1)]$) depends on $u(H, t)$, the observed average fitness of H .

However, this is only one side of the story. Crossover and mutation are also part of GA and these can both destroy as well as create instances of H . For a conservative estimate of the growth of instances of H we consider only the destructive effects of crossover and produce a lower bound on $E[m(H, t+1)]$ as follows. If p_c is the probability that a given string instance of H will experience a single point crossover, then H will survive under single point crossover if one of the offspring of such crossover is also an instance of H . The lower bound on the probability of such survival may be given, under random selection of the crossover site, as

$$\text{Survival Probability of } H \text{ under crossover} \geq 1 - p_c d(H) / (l - 1)$$

when $d(H)$ is the defining length (the distance between the two outermost *defined* bits of H) and $(l - 1)$ is the number of possible one-point crossover sites in a string of length l . Note that given p_c , this survival probability will be high for the schemas with small defining length.

The disruptive effect of mutation may also be estimated and accounted for. If p_m represents the probability of any bit being mutated, then the probability that the schema H will survive a mutation is $(1 - p_m)^{o(H)}$, where $o(H)$ is the order of (that is the number

of defined bits in) H . This implies that the probability of surviving a mutation is higher for lower-order schemas.

The disruptive effects of crossover and mutation may be now combined. We may write

$$E[m(H, t+1)] \geq [m(H, t) u(H, t) / \mathcal{F}_{avg}(t)] [1 - p_c d(H) / (l-1)] (1 - p_m)^{o(H)}$$

The above result is a fundamental contribution made by Holland and is known as Holland's Schema Theorem. The practical implication of the schema theorem is that short, low-order schemas whose average fitness remain above the population average fitness $\mathcal{F}_{avg}(t)$ will receive exponentially increasing numbers of instances over time.

2.7.4 The Building Block Hypothesis

Notice that the right hand side of the above result gives us a lower bound because we dealt here with only the destructive effects of crossover and mutation. In actual fact, of course, crossover is believed to be a major source of the GA's power, rather than a weakness.

Crossover is an attempt to produce equally good or higher order schemas by recombining instances of good schemas. Goldberg (1989) was the first to put forth this supposition about the behavior of GAs. He called it the "building block hypothesis."

Recall also that we used the binary coded scheme and the single-point crossover as the premise for the above derivation. Many other types of coding methods and the corresponding crossover and mutation schemes have now been devised and implemented, not only in numerical optimization using real-valued variables directly, but also in combinatorial optimization such as solving the traveling salesman problem, vehicle routing, communications code design and many machine scheduling and sequencing problems (Mitchell, 1996; Aarts and Lenstra, 1997).

2.8 SCHEMA PROCESSING: AN ILLUSTRATION

The processing of schemas may also be illustrated by the two-variable optimization example given earlier. Consider, for example, the growth of a particular schema $H = (0 * \dots * \dots *)$. Note that this schema represents all solutions with x_2 bounded in the region $0 \leq x_2 \leq 3$. Actually, the second substring of H decodes into a value $\leq 2^9$. This implies that x_2 would have a value $\leq [0 + (6 - 0) 2^9]/1023$ or ≤ 3.0 . (The optimum point lies in this region.) The order of this schema is 1, and its defining length is also 1. The function values of solutions represented by H would be at most $(0 + 3 - 11)^2 + (0 + 3^2 - 7)^2$ or 68. This makes its fitness \mathcal{F} at least $1/(1 + 68)$ or 0.014493. Note that this fitness value is greater than 0.008, the F_{avg} of the initial population. This makes H a *building block* schema. The competitor of H is the schema H^c , which is $(0 * \dots * \dots *)$. H^c represents solutions in the region $3 \leq x_2 \leq 6$ and therefore has a lower fitness \mathcal{F} . By the schema theorem, the appearance of schema H should therefore increase in the population as the generations evolve and the reverse should be true for H^c . The count of H in the initial random population is 9 whereas that of H^c is 11. At the end of one generation, H is represented by 14 strings while H^c is represented only by six strings. This illustrates the growth of low-order, above average schemata as the GA iterates.

Other schemata with low order and high fitness would also propagate in the population in the similar manner. According to Goldberg (1989), such processing occurs in parallel, without explicit bookkeeping of the solutions found.

2.9 ADVANCED MODELS OF GENETIC ALGORITHMS

Modeling natural evolution mathematically and then applying this knowledge to draw benefits in optimization is a challenging task. While practical methods may be inspired by watching experienced animal and plant breeders, evolutionary algorithms (including the GA) are actually mathematical random search methods, best probed by modeling.

One approach to model the GA would be to use population genetics. Such studies would help in addressing the following questions. How should selection be done, given a selection and crossover scheme?

What is the expected fitness improvement of the population, given crossover and mutation? How can selection, mutation and crossover be combined in a synergistic manner to achieve convergence? A good model would perhaps express the rate of convergence to the global optima as a function of the algorithm's operating parameters such as population size, mutation and crossover probabilities, and exploratory information about the *fitness landscape* (the nature of objective function being optimized).

Holland's analysis which culminated in the schema theorem actually took the opposite stand. The schema theorem predicts the progress under a specific selection scheme known as proportionate selection. Mutation and crossover (also known as recombination) are later introduced as disturbances (see Section 2.7.3). Recently, other experts have cited the role of mutation and crossover being *constructive*. A single mutation step indeed is almost unpredictable, for mutation is based on chance. A crossover, on the other hand, is an attempt to perform a global search based on restricted chance. A crossover only shuffles substrings contained in the population. Therefore, in order for a crossover to locate the optimum, the substrings of the optimum solution have to be present in the population. To be effective, therefore, a crossover needs a large population size.

GAs actually trace a complex stochastic process. It is also observed that the choice of parameters such as the probability of crossover (p_c), the probability of mutation (p_m), the nature of the function or response being optimized, the type of crossover and mutation schemes implemented, etc., *interact* with each other in a complex manner to determine the algorithm's convergence. Broadly speaking, two types of convergence are sought in an algorithm. *On-line convergence* is a measure of how rapidly the algorithm improves the average quality of the solutions. *Off-line convergence*, on the other hand, is the capability of the algorithm to consistently reach the true or global optima rather than getting trapped at some local optima on the way. For GA to be effective, robust on-line and off-line performance are both desirable.

Convergence being a primary concern, a number of different approaches to model the GA process have been recently attempted. Some researchers have used the analogy with the two-armed bandit problem (Mitchell, 1996) while others have attempted to model simple GAs as Markov chains (Vose and Liepins, 1991). Mitchell (1996) summarizes these approaches and states their limitations and

strengths. Yet another model of evolutionary algorithms based on population genetics is provided by Muhlenbein (1997). Doubtless, if an analytical model for the GA's stochastic search can be constructed, one may "parameterize" the process to make it most effective as we often do in electrical engineering. For GA, the analytical work on population genetics by Muhlenbein (1997) has already led to the creation of a novel and effective crossover scheme that he calls *gene pool crossover*. Analytical models, therefore, would save us a great deal of the trial-and-error or experimental effort currently expended. However, till the ultimate model is built for GA (as is also the case in chemical or metallurgical process scale up, product design, design of FMS configuration, etc.), we have to derive *empirical laws*. Empirical laws may come from carefully performed physical or computer experiments (simulation).

For the use of GAs in practical optimization problem solving, the key question that still remains open is the following: Can the optimal individual solution be reliably discovered by GAs in a reasonable period of time?

Two approaches are used by GA practitioners to gain speed. The first is to parameterize the GA empirically to maximize the rate of convergence while ensuring its robust performance. A good deal of work has already been done here and we describe some of these in Chapter 3 of this text. In that chapter we describe a statistical design-of-experiments approach to parameterize GAs to ensure their fast and robust convergence in routine applications, regardless of the starting condition. The second approach is to hybridize. One possibility here is to use GA to find a "reasonably good" rather than the globally optimal solution—in a reasonably short time. Mitchell (1996) calls this the "satisficing" approach. The GA solution thus produced may be then used as the starting point of some conventional optimization method, such as hill climbing. This *hybrid* method is used in many applications it is reported that such a solution strategy performs better than a GA alone. Another productive way to hybridize a GA is to incorporate Lamarckism in it (Morris et al., 1998). We shall see such an application in Section 4.8.

In the author's own experience and that of many others (see, for instance, Uckun et al.), if the GA can be assisted in its execution by augmenting it with *problem domain-specific knowledge*, the quality of the final solution frequently improves. Also, when constraints are

present, appropriate solution coding schemes can greatly enhance the search process. This works very well, for instance, in classroom scheduling or school timetabling by GA. When suitable *solution-improvement methods* are available, these may be used to assist in some or all solutions in *each* GA generation. The Ho and Chang (1991) heuristic for flowshop sequencing is one such aid. Other such methods include the shifting bottleneck heuristic by Balas and Vazacopoulos (1994) for the job shop. Similar improvement methods also exist for the traveling salesman problem.

Still other ways are being continually experimented with and proposed. These include the "micro-GA" (Krishnakumar, 1989), a very effective GA formulation that works with a very small size population, often only 5, for its execution. GAs have also been used with niching now to tackle *multiobjective* problems. It has been shown that GAs can often develop Pareto optimal solutions in such situations very effectively (Srinivas and Deb, 1995).

Nevertheless, many key questions about GAs still remain open, such as what is the expected time to find a particular fitness level or what makes a problem hard for GA. Also, a GA can spend a considerable amount of time without showing improvement, and then suddenly produce a good jump. This characteristic appears to be a function of the degree of mutation used. Such phenomena cannot be predicted ahead of time. When available, the theoretical answers to such questions would enable us to determine what GA is good at doing and how to make it more effective. The two other useful metaheuristic methods are simulated annealing (Metropolis et al., 1953), which has a convergence proof, and tabu search (Glover et al., 1997).

Many use the GA "because it works; it produces solutions" even if we do not have a proof for its convergence except for some simple and special fitness landscapes. Hence its effective use remains essentially an empirical enterprise. Still, its easy intuitive appeal and the large increase in computational resources with sophisticated data structures sustains its popularity in solving a large variety of real life optimization problems including engineering design optimization problems (see Grierson and Hajela, 1996). Be that as it may, the *population-based* structure of the GA, at least for the time being, would sustain its use in situations where multiple solutions may be sought. One such domain is the determination of Pareto optimal solutions to multiobjective machine scheduling problems—the subject of this text.

3

CALIBRATION OF GA PARAMETERS

This chapter focuses on a critical dilemma faced in many GA applications: the optimum selection of the different GA parameters to ensure the GA's rapid convergence—both *on-line* and *off-line* (Section 2.9). We address this task early in this text because the satisfactory resolution of it can either make or break the efficacy of the GA, for there is no single makeup of a GA that can uniformly solve *all* global search problems. In this respect the GA is unlike, for instance, the Newton-Raphson method which finds zeros of almost any polynomial. In particular, the stochastic process that develops as the GA is executing is noted to be affected by the values chosen for the *elite fraction* (ϵ) participating in reproduction, *crossover probability* (p_c), *mutation probability* (p_m), *population size* (p_s), etc. The GA's convergence may be impacted also by the *interaction* among these parameters. However, no general methodology is presently available to optimize the selection of these parameters. What is even more troublesome is the growing evidence that such "optimum" parameter values may be problem-specific. This chapter presents a robust parameterization procedure based on the statistical design of experiments (DOE) approach (Bagchi and Deb, 1996). A multi-factor constrained optimization problem with known solution is used to illustrate the steps in this proposed method.

3.1 GA PARAMETERS AND THE CONTROL OF SEARCH

Soon after De Jong (1975) extended John Holland's inspiring ideas to seek global maxima or global minima of complex functions using GAs, it became clear that Holland's schema theorem was by itself insufficient to guide the effective use of the GA meta-heuristic in global optimization. Others including Muhlenbein (1997) also point

this out. The schema theorem does not indicate what specific values one should choose, for instance, for the elite fraction (ε) of the population to be used in reproduction, the crossover probability (p_c), the number of crossover points, population size (p_s), the mutation probability (p_m), and also the different schemes that introduce a controlled amount of randomness and/or local search while the GA is executing. However, computational experience with GA quickly indicates that the GA's efficiency and also its effectiveness in avoiding getting trapped in local optima depend to a great degree upon the selection of these "control parameters."

GA search for the global optima should ideally strike a good balance between the *exploration* of the total search space and the *exploitation* of good solutions currently at hand. De Jong's Ph D thesis (De Jong, 1975) perhaps was the first systematic attempt to seek out optimum settings for GA parameters. Another celebrated work is that of Grefenstette (1986), who remarked that it would not be easy to predict, for instance, the effect of increasing population size, while lowering crossover rate, because the effects of these parameters may interact. Furthermore, it is now widely held that "optimum parameter settings" may be problem-specific, suggesting that one must first parameterize the GA in the context of the particular problem of interest. Only then should one mount a full-scale effort to optimize that problem using GA.

In the following pages we first recall the importance of GA parameterization as noted by recent researchers. Next we present a parameterization method that is based on the principles of statistical design of experiments (DOE, Montgomery, 1996). We then show how DOE quickly and efficiently locates the best GA parameter settings for the optimization problem at hand. Based on its utility demonstrated in optimization problems ranging from Taguchi-type robust process design to shop scheduling, we conclude that the use of DOE in selecting p_s , p_c , p_m , etc., is clearly more rewarding than the use of parameter values chosen ad hoc.

The impact that ε , p_s , p_c and p_m , etc. can have on the algorithm's convergence toward the optimum solution is well documented (see, for instance, De Jong, 1975, Grefenstette, 1986; Davis, 1989; Schaffer et al., 1989; Davis, 1991; or Srinivas and Patnaik, 1994). Grefenstette (1986) used the GA itself at a metalevel to search for good values for

crossover and mutation probabilities. Parameterization has also been attempted by exhaustive search entailing extensive computation (Murata et al., 1996). Davis (1991) demonstrated that the difference between using randomly picked parameter values rather than optimum values can easily cause convergence to be delayed by "an order of magnitude." Many other recent studies have tried to identify optimal parameter settings. Still, no theoretical prescription is available as yet to specify these settings and the guidelines at hand are either certain rules-of-thumb or isolated empirical evidence. Consequently, the debate continues over whether such settings are unique for each problem, or are they robust? Or should they be dynamically tuned as the genetic search progresses? Reflecting on the extent of disturbance that is caused to good schemata as the GA search progresses, Davis (1991) suggested four different methods for "adaptive" parameterization to ensure good convergence. Srinivas and Patnaik (1994) give another adaptive scheme. In their "Adaptive GA," low values of p_c and p_m are assigned to high fitness solutions as they are discovered while low fitness solutions receive very high values of p_c and p_m . For highly epistatic and multi-modal problems tested, Srinivas and Patnaik report their scheme doing better than the "Simple GA" operating with constant p_c and p_m . In the next section we obtain a glimpse of the theoretical bases for seeking optimum parameter settings. We then explore parameterization using the design of experiments (DOE) approach.

Aside from parameterization, another important consideration that impacts genetic search is the representation structure—also known as "coding." Representation defines genetic operators, which would have the required characteristic of passing some characteristic structure from parent to offspring. This is noted to be particularly important in scheduling applications (Bruns, 1993; Kumar, 1997). The representation structure should be such that the schemata are meaningful and relevant to the problem being solved and that the GA operators are able to juxtapose at will the underlying building blocks to enable the search process to progress.

DOE is an investigative method regarded highly both for its effectiveness and its efficiency in evaluating the effect of multiple factors upon a process. Its theoretical foundations are sound and its application in multi-factor cause-response studies has been rigorously developed (see, for instance, Cochran and Cox, 1957, or Montgomery,

1996). Use of DOE to parameterize GAs would therefore seem natural.

The most serious limitation of the existing parameterization methods is that there is no explicit consideration in them to take care of *interaction*. Two factors (for instance, two GA parameters) are said to interact if the observed effect of one factor (e.g. the role it plays in speeding up or slowing down the GA's convergence) depends on the level of the *other*. Also, rarely is the GA efficacy attributable to a single parameter. We recall that DOE can find factor effects singly, as well as it can uncover any significant interaction that may exist among the effects attributable to these factors.

One objective of this chapter is to also illustrate the potential utility of DOE as a *general methodology* for parameterization of algorithms, be they GAs or otherwise, an aspect which appears to have been overlooked by many users of these algorithms. The investigative schemes (full factorials, orthogonal arrays, Latin square designs, etc.) that guide DOE are based on sound statistical theory. The associated data analysis method (ANOVA) is precise (Cochran and Cox, 1957). However, we caution that the specific parameter values arrived at using DOE are normally context-specific, and therefore the settings thus identified may not have general validity in *all* GA applications. This is so because the values of GA parameters may interact in a complex manner with the structure of the solution space at hand in determining convergence. However, in the studies presented in this chapter DOE has been tested as a parameterization methodology across a broad range of GA applications—to evaluate its robustness as a *technique*.

We emphasize here that we propose the use of DOE as the *parameterization methodology* (till reasonably complete stochastic models of the GA evolution process are developed) and not the use of the particular values for ε , p_s , p_c or p_m found in the illustrations given in this chapter as "cure all" values for these parameters in all GA applications. Therefore, presentation in this chapter is illustrative rather than exhaustive. A job shop scheduling study using DOE would include, for instance, multiple crossover operators, crowding considerations, different selection schemes, etc. as the DOE factors. Nonetheless, we feel that DOE-based parameterization would be a productive stride not only for GAs, but also for simulated annealing and other algorithmic methods requiring parameterization.

In solving a numeric optimization problem using GA, one utilizes a suitable representation structure and then selects the string length l . The choice of l is largely based on the precision desired in the solution and on the number of independent design variables involved. In general, for binary-coded GAs the precision expected of a string of size l , as Deb (1993) indicates, is of the order

$$O((x_{\max} - x_{\min})/2^l)$$

where x_{\max} and x_{\min} are the extreme points of the continuous search space along variable x . Goldberg et al. (1993a and 1993b) proposed control maps for the values of the selection pressure ς (the number of copies allocated to the best string in the population) against the crossover probability p_c (the extent of search), for bit-wise linear problems. They show that in order for GAs to be effective the following condition should be satisfied:

$$p_c \geq \frac{e^{l/p_s}}{p_s \log p_s} \log \varsigma$$

where p_s is population size. An alternate strategy is called "steady-state without duplicate," which avoids duplication of individuals in the population to make more efficient use of the allotted number of individuals (Davis, 1991). Earlier, Goldberg (1985) had derived a count of unique expected schemata for strings of given length and populations of a given size. There he provided a rational and problem-independent performance measure (the number of excess schemata per population member) for selecting optimal *population sizes*. This performance measure is maximized using Fibonacci search and Goldberg presents the results in tabular as well as graphical forms. An empirical fit of this numerical result provides an approximate relationship between the optimal population size (p_s) and string length (l) as

$$p_s = 1.65 \times 2^l$$

valid for string lengths up to 60. Grefenstette (1986) suggested that a high crossover rate tends to disrupt the structures selected for reproduction at a high rate, important in a small population. The

following section provides a glimpse of the theoretical reasons as to why care must be exercised in choosing the settings of the different GA parameters.

3.2 THE ROLE OF THE "ELITE" WHO PARENT THE NEXT GENERATION

As mentioned in Chapter 2, Darwin in 1859 proposed that natural evolution favors those members of the species that are better adapted to their environment and it tends to eliminate those that are less so. These "fittest" organisms can form the "elite" in the population who survive and procreate. As the result, through successive generations changes become established that lead to the production of new species with improved adaptation. GAs seek progeny of higher fitness (Goldberg et al. 1993a, 1993b) by emulating what nature is believed to do through natural selection, recombination and mutation to adapt organisms to their environment. For details on the mathematical analog of natural adaptation as contrived by GA we refer you to Holland's classic text on the subject (Holland, 1975).

However, it is not certain whether nature "optimizes" its processes; in fact, it probably doesn't (Smith, 1993). Nature's priority apparently is local adaptation rather than speed. In applying GA to real optimization problems, though, we would be strongly interested in a *rapid* convergence to the global optimum (i.e., good on-line performance), while also avoiding getting trapped in the local optima (good off-line performance). Still, not only GAs, but also the efforts under way in genetic engineering would benefit much if the time to reach certain desired ends could be shortened by optimally balancing say *population diversity* and *selective pressure*. Indeed we may create efficient GAs if we first understand clearly the interacting elements of the different processes that nature seems to have contrived to evolve the living world. In this section we probe the effect of selective pressure on the average fitness of progeny.

As noted earlier, *selection* aims at retaining those strings (for reproduction, to create the next generation) that have a higher fitness, to lead to an even higher level of average fitness of the progenies. Several different selection schemes are in use, a popular one being *proportionate selection* (Grefenstette, 1986). Proportionate selection

selects the new population (by reproducing the parents) in proportion to a probability distribution based on the fitness of the individual parents. We derive below the growth of average fitness in a *single* application of proportionate selection and reproduction, accomplished with a set of parent strings tagged as "elite" because of their above average fitness at time t .

Suppose that the objective is to maximize the function $\phi(x) = x$ in the domain $0 \leq x \leq 1.0$, using an abbreviated GA given by

```

procedure maximize  $\phi$  ;
begin
     $t \leftarrow 0$  ;
    initialize population  $P(t)$  by randomizing ;
    evaluate population  $P(t)$  ;
    while (termination criteria not met) do
        begin
             $t \leftarrow t + 1$ 
            select elite ;
            proportionate reproduction ;
            evaluate population  $P(t + 1)$  ;
        end ;
    end ;

```

Suppose that X_1, X_2, \dots, X_p form an original population $P(t)$ of p strings ("Generation 0"), randomly selected from the interval $[0, 1]$, at time $t = 0$. Suppose further that their fitness is unequal, and when arranged in the nondecreasing order of their fitness, they form the *ordered* sequence

$$Y_1, Y_2, Y_3, \dots, Y_{p-\varepsilon}, Y_{p-\varepsilon+1}, \dots, Y_p$$

such that $\text{eval}(Y_{i+1}, 0) \geq \text{eval}(Y_i, 0)$ when $\text{eval}(Y_i, 0)$ represents the value or fitness of string Y_i at generation 0. To create the next generation (for $t = 1$) we use the *best* ε strings, identified as members of the "elite set" $\{Y_{p-\varepsilon+1}, \dots, Y_p\}$. In the proportionate reproduction scheme, members of the elite set are reproduced in proportion to their average fitness, to reconstitute the population back to its original size p . How would the *average fitness* $P(t = 1)$ of the progeny ("Generation 1") change from that of the initial generation ("Generation 0") in this process?

Note that in the abbreviated GA shown above, population size (p) and the size of the elite group (ε) are the only two control parameters.

Since the initial population $\{X_i\}$ is selected randomly in the range $[0, 1]$, the values of X_i would be uniformly distributed in $[0, 1]$. Clearly, the expected fitness of $\{X_i\}$ will be

$$\mu_{\text{generation } 0} = 0.5$$

A standard result in multivariate distribution theory is as follows. Let X_1, X_2, \dots, X_n be n independent random variables with a common distribution function F . If we arrange $\{X_i, i = 1, \dots, n\}$ in increasing order represented by Y_1, Y_2, \dots, Y_n , then $\{Y_i, i = 1, \dots, n\}$ are called the *order statistics* of $\{X_i\}$. Now, if F has a piecewise continuous density f , then for $1 \leq k \leq n$, Y_k has the density g_k , where

$$g_k(y) = n \frac{(n-1)!}{(n-k)!(k-1)!} F(y)^{k-1} [1 - F(y)]^{n-k} f(y) \quad (3.1)$$

for $-\infty < y < \infty$ (Montgomery and George, 1998). The application of this result to the present problem is straightforward. The distribution of each X_i here is uniform in $[0, 1]$ and $\{X_i\}$ are independent, hence $F(X_i) = F(X) = x$ and $f(x)$ is 1. Therefore, we get

$$g_k(y) = p \frac{(p-1)!}{(p-k)!(k-1)!} \frac{y^{k-1} (1-y)^{p-k}}{p}$$

$g_k(y)$ gives the distribution of the k^{th} member in the ordered set $\{Y_i\}$, called the k^{th} order statistic of $\{X_i\}$. The distribution given by (3.1) is a standard probability distribution, known as the beta distribution. The mean (μ_k) and the variance (σ_k^2) of this distribution are given by

$$\mu_k = k/(p+1)$$

and

$$\sigma_k^2 = k(p-k+1)(p+2)/(p+1)^2$$

In our case, since we have selected ε elite (the top ε values of Y_i) out of the ordered set $\{Y_i, 1 \leq i \leq p\}$, the average fitness of these ε elite would be

$$\begin{aligned}\mu_{\text{elite}} &= \frac{p}{\sum_{k=p-\varepsilon+1}^p \mu_k / \varepsilon} \\ &= 1 - (\varepsilon + 1) / [2(p + 1)]\end{aligned}\quad (3.2)$$

It may be quickly verified that if *all* members of the ordered set $\{Y_i, 1 \leq i \leq p\}$ are selected to be elite, then $\varepsilon = p$, and consequently $\mu_{\text{elite}} = 0.5$. And, if only the top member of $\{Y_i\}$ is made the sole elite, then $\mu_{\text{elite}} = p/(p + 1)$, which coincides with the mean of the p^{th} ordered statistic Y_p .

The next step in the abbreviated GA is to perform proportionate reproduction using the ε elite $\{Y_i, i = p - \varepsilon + 1, \dots, p\}$. According to this scheme, the probability of selecting some particular elite string Y_i would depend on its relative fitness (ϕ_i). This probability is proportionate to ϕ_i and is given by π_i when

$$\pi_i = \phi_i / \sum_{j=1}^{\varepsilon} \phi_j$$

Now, the fitness of the strings $\{Y_i, i = p - \varepsilon + 1, \dots, p\}$ is proportional to their respective means. Hence, we have $\phi_i = \text{mean}(Y_i) = \mu_i$. It may be therefore shown that

$$\pi_i = i / [p + 1 - (\varepsilon + 1)/2] \varepsilon, \quad i = 1, \dots, \varepsilon$$

$\{\pi_i, i = 1, \dots, \varepsilon\}$ give the probabilities of selecting the corresponding elite from $\{Y_i, i = p - \varepsilon + 1, \dots, p\}$. It is easy to verify that $\sum \pi_i = 1.0$. Restoration of the size of the population back to p by proportionately reproducing the elite using $\{\pi_i\}$ completes reproduction.

Our own objective here is to estimate the average fitness of Generation 1, the population reconstituted after proportional reproduction. With some algebra it may be shown that

$$\begin{aligned}
\mu_{\text{generation_1}} &= \frac{2}{\varepsilon(p+1)(2p-\varepsilon+1)} \sum_{j=1}^{\varepsilon} (p - \varepsilon + j)^2 \\
&= \frac{6p^2 - 6p\varepsilon + 6p + 2\varepsilon^2 - 3\varepsilon + 1}{3(p+1)(2p-\varepsilon+1)} \tag{3.3}
\end{aligned}$$

We have finally derived the expression (3.3) through which we would be able to estimate the impact of proportionate reproduction of elite constituted by selecting the top ε best performing initial (Generation 0) solutions $\{X_i, i = 1, \dots, p\}$. Again, it is not difficult to verify from (3.3) that if only the single best elite is selected for reproduction to reconstitute the population in Generation 1, then $\mu_{\text{generation_1}} = p/(p+1)$, which coincides with the mean of Y_p . Note that $\mu_{\text{generation_1}}$ is a decreasing function of ε , the size of the elite group. This is not surprising: the fewer the number of the top elite selected to reproduce, the higher will be the average fitness of the proportionately reproduced progeny. Also, $\mu_{\text{generation_1}}$ is an increasing function of population size p . This effect is perhaps not obvious immediately. Figure 3.1 displays these relationships graphically for $p = 100$ and elite count ε varying from 1 to 50.

Note that the lines drawn on Fig. 3.1 are not parallel, evidencing the presence of *interaction* between the effects of p and ε on $\mu_{\text{generation_1}}$. The upshot of this reality is that to maximize $\mu_{\text{generation_1}}$ one must simultaneously optimize control parameters p and ε ; their effect is *not* additive (Montgomery, 1996). We must note that the above deduction is not empirical. It is based on probability theory. Further, for some other fitness function $\phi(x)$ other than $\phi(x) = x$, we are likely to have an even different relationship between $\mu_{\text{generation_1}}$, p and ε , obviating the need to parameterize the abbreviated GA in *that* domain. We must quickly add that working with the abbreviated GA without some means to introduce new schemata into the population *will not improve the solution*. The best solution achievable here is already present in Generation 0! That is why we need to employ crossover (recombination) and mutation, in addition to selection, in a GA. Selection pressure, nevertheless, would speed up search.

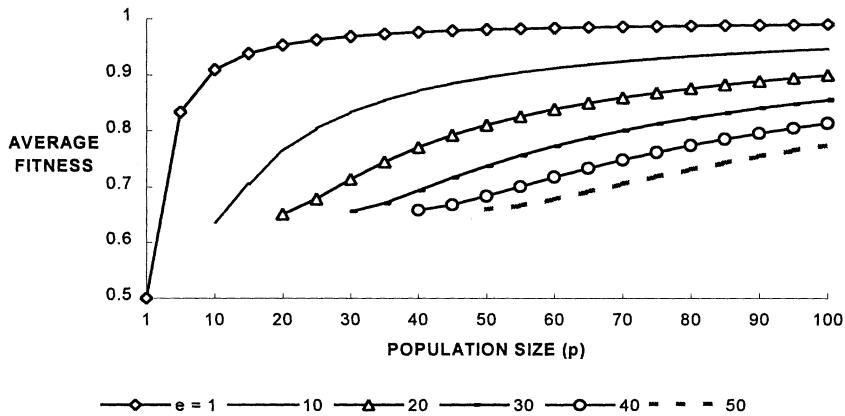


FIG. 3.1 AVERAGE FITNESS OF GENERATION 1 BY PROPORTIONATE SELECTION OF ELITE

When randomness is introduced through recombination (crossover) and mutation, the prediction of $\mu_{\text{generation_1}}$ and the fitness of the later generations rapidly becomes analytically difficult, even intractable. Since parameter optimization would then involve multiple parameters, it would seem natural to study the problem empirically, using a DOE-type framework. This is the approach described in the remainder of this chapter.

The choice of the "test bed" (a test problem or a class of problems) to conduct the parametric studies has been already observed to influence the values of the parameters that are identified as "optimum" (Grefenstette, 1986; Whitley, 1989 and Davis, 1991). In other words, these values may not be globally optimum or robust, i.e., valid for *all* GA-based search problems. Therefore, we would regard any parametric study as a *calibration* rather than as an optimization study. Therefore, it is important that similar calibration of the parameters be done in a pilot study in the target problem domain, especially when the optimization problem at hand has some unusual character.

3.3 THE FACTORIAL PARAMETRIC STUDY

The object of calibrating (loosely said "optimizing") the GA parameters— p_s , p_c , and p_m —would be to identify the values of these

parameters for problems involving a fixed string length (l) in order that an objective function value close to the true optima may be reached by the GA search, and that this may be done using the *smallest* number of function evaluations. It is very possible, as we show here, to use the factorial design framework (see Cochran and Cox, 1957) to empirically identify such optimum combination of these parameters.

3.3.1 The Experimental Layout

The optimization of the parameters p_S , p_C , and p_M itself is a global search problem: it is a multifactor problem and one *without* a known response structure. Such problems are not new, however. These may be investigated using an appropriate experimental design scheme (Montgomery, 1996). The layouts or experimental schemes possible here are numerous, ranging from the "full factorial design," to the simple "orthogonal array (OA) design," all involving the experimental factors and a response or test function. The selection of experimental layout is determined by the objective of the investigation and by the cost of conducting the experiments.

To illustrate the use of factorial design in GA parameterization we shall use here an objective function provided in a paper on robust design by Vining and Myers (1990) as the test function. Robust design is a powerful approach to create product and process designs that retain their performance under diverse environmental conditions. Robust design presents a wide class of challenging global optimization problems and its details are given in Phadke (1989) or Bagchi and Templeton (1994).

The Vining-Myers problem required the minimization of the standard deviation (σ) given by

$$\begin{aligned}\sigma = & 34.9 + 11.5x_1 + 15.3x_2 + 29.2x_3 + 4.2x_1^2 - 13.2x_2^2 \\ & + 16.8x_3^2 + 7.7x_1 x_2 + 5.1x_1 x_3 + 14.1x_2 x_3\end{aligned}\quad (3.4)$$

subject to the constraint on mean ($\mu = 160$)

$$\begin{aligned}\mu = & 327.6 + 177x_1 + 109.4x_2 + 131.5x_3 + 32x_1^2 - 22.4x_2^2 \\ & - 29.1x_3^2 + 66x_1 x_2 + 75.5x_1 x_3 + 43.6x_2 x_3\end{aligned}\quad (3.5)$$

when the three (robust design) decision variables are bounded in the range

$$-1 < x_1, x_2, x_3 < +1 \quad (3.6)$$

We use the above problem as the "test bed" problem because this problem's degree of nonlinearity is explicitly known and because there would virtually be no distortion produced by the relevant penalty function as there are many potential solutions present in the region bounded by (3.6) that satisfy (3.5). Further, and importantly for this illustration, the global solution to this particular problem is known (see Castillo and Montgomery, 1993), providing us with a known benchmark for comparing the performance of the GA at various settings of p_s , p_c , and p_m .

To parameterize the GA to solve the above problem efficiently, the problem was converted first into an unconstrained minimization problem, by eliminating the variable x_1 from the target constraint. The boundary constraints (3.6) were handled by incorporating a penalty term in the objective function. A string length of 15 per design variable was used—to provide a precision of the order of 0.0001, making up a combined string of length (l) 30 for the two remaining search variables x_2 and x_3 . Binary type coding was used throughout.

The effectiveness of binary coding was noted as early as Holland's original work in 1975. Spears and De Jong (1991) write that binary or two-valued genes each take on a small number of values and are thus in a sense optimal for GA-style adaptive search. GAs perform significantly better with binary representation because then, in addition to mutation, crossover generates new decision variable values each time it combines part of a decision variable's bits from one parent with those of another.

The value of binary coding may be illustrated as follows. Consider the extreme case in which we must adjust a single decision variable represented by a chromosome with *a single gene* that can take say 2^{15} distinct values. Recall that the role of crossover is to variously juxtapose the mating parents' *whole genes*. Therefore, representing this problem as a one-gene problem (the single decision variable being represented completely by a single gene) would make crossover

useless and leave mutation as the only mechanism for generating any new individuals. However, here a *15-gene binary representation* would let crossover play an active and crucial role as well, with high performance expectations.

The actual experiments were "pilot" runs of the GA lasting for a *fixed*, limited number of generations seeking to minimize σ . Each such pilot GA run was parameterized precisely as guided by the p_s , p_c and p_m values shown in the different rows of Table 3.1. The pilot runs themselves were of short duration, each running for a given number (= 4000) of function evaluation, to enable detection of the parameter value combination showing the best rate of convergence.

**TABLE 3.1 THE 3^3 FULL FACTORIAL LAYOUT SHOWING
FACTOR LEVEL COMBINATION FOR EACH OF
THE 27 PILOT GA EXPERIMENTS**

Expt #	p_s	p_c	p_m
1	0.7	0.50	0.6/
2	0.7	0.50	1/
3	0.7	0.50	3/
4	0.7	0.75	0.6/
5	0.7	0.75	1/
6	0.7	0.75	3/
7	0.7	0.95	0.6/
8	0.7	0.95	1/
9	0.7	0.95	3/
10	1.3	0.50	0.6/
11	1.3	0.50	1/
12	1.3	0.50	3/
13	1.3	0.75	0.6/
14	1.3	0.75	1/
15	1.3	0.75	3/
16	1.3	0.95	0.6/
17	1.3	0.95	1/
18	1.3	0.95	3/
19	2.7	0.50	0.6/
20	2.7	0.50	1/
21	2.7	0.50	3/
22	2.7	0.75	0.6/
23	2.7	0.75	1/
24	2.7	0.75	3/
25	2.7	0.95	0.6/
26	2.7	0.95	1/
27	2.7	0.95	3/

In the experiments we set the three parameters— p_s , p_c and p_m —each at *three* levels, to detect any nonlinear effects. Further, in order to permit the detection of the possible interaction of factor effects, we used a 3^3 full factorial experimental layout shown in Table 3.1 (see Cochran and Cox, 1957 or Montgomery, 1996). Note again that each "experiment" was a GA run using the test problem as the "subject," the convergence of the GA search being tracked in each case till a stopping condition was reached. Table 3.1 shows the actual levels used for p_s , p_c and p_m (expressed in terms of string length l) which would cover a broad region of the total parameter space. These values are indicative of the "working ranges" that might be otherwise selected for the three GA parameters tested. Quite naturally, if the DOE exercise found "optimum" setting to be at one of the extremes, another set of experiments would be run.

We must also recall (see page 29 of Davis, 1991) that since a GA *repeats* its performance on a given problem, we would need multiple runs—at each parameter setting—to determine that a variation (a different combination of parameter values) performs indeed differently on the subject problem. Davis notes that a single GA run could lead to a lucky hit or an improbable miss that giving misleading information about the hypothesis one is testing (that a given parameter setting is the best, for instance). In the present evaluation "noise" was deliberately introduced to avoid such "lucky hit/miss."

The random seed used in initializing program execution was varied under control. For each experimental "run" of Table 3.1, five replicate observations were made by changing the random seed that initialized the GA. Further, to improve statistical precision, the notion of variance reduction (Law and Kelton, 1991) was used: the random seeds were kept identical for every row in Table 3.1, each row being a "run" with pre-set p_s , p_c and p_m values.

The GA we used was a "tripartite GA," i.e., one with three operators, namely, binary tournament reproduction, single-point crossover, and bit-wise mutation (Goldberg, 1989). The stopping criterion used was a fixed number (4000) of function evaluations for all runs. For different combinations of p_c , p_m and N_g (N_g being the different number of generations for which the program would run) such a criterion would imply that

expected number of function evaluations

$$= p_s p_c N_g = 4000$$

Thus, if $p_s = 80$ and $p_c = 0.5$, then the GA would be expected to evolve for ($N_g =$) 100 generations on an average. Table 3.2 displays the lowest sigma (σ) value obtained in each of the replicated GA runs parameterized in accordance with the values for p_s , p_c and p_m as shown against the corresponding experiment in Table 3.1.

TABLE 3.2 "BEST" OBJECTIVE FUNCTION VALUES OBTAINED IN EACH REPLICATED RUN OF THE FACTORIAL EXPERIMENTS OF TABLE 1

Replication # →	1	2	3	4	5
Expt #1	45.2358	45.5877	45.6045	45.7479	45.8655
2	45.2219	45.1893	45.6869	45.2242	45.8459
3	45.1278	45.9357	45.2217	45.2607	45.2803
4	45.1630	45.1905	45.9190	45.2588	45.1054
5	45.8799	45.2142	45.7348	45.2100	45.2841
6	45.1270	45.2930	45.1180	45.1446	45.1306
7	45.1054	45.7888	45.3343	45.5727	45.2840
8	45.1076	45.6158	45.2697	45.5273	45.0996
9	45.3275	45.1677	45.1182	45.1035	45.1226
10	45.1622	45.9260	45.5856	45.8827	45.1325
11	45.1622	45.9260	45.5886	45.8827	45.1325
12	45.2107	45.2541	45.1728	45.1452	45.2122
13	45.4011	45.6616	45.1963	45.2173	45.7683
14	45.3210	45.3065	45.1963	45.3921	45.7832
15	45.1066	45.1317	45.3798	45.1419	45.2189
16	45.1171	45.8709	45.1771	45.6674	45.2728
17	45.1848	45.3119	45.1077	45.3010	45.1740
18	45.1046	45.1010	45.1163	45.1085	45.1676
19	45.1098	45.2003	45.8483	45.2549	45.2549
20	45.1054	45.1307	45.0998	45.1160	45.1753
21	45.1490	45.1083	45.1263	45.1057	45.0995
22	45.2626	45.3561	45.1694	45.1051	45.3451
23	45.1402	45.1309	45.1162	45.1393	45.1254
24	45.1275	45.0999	45.1140	45.1097	45.1027
25	45.1616	45.1792	45.1134	45.1639	45.1192
26	45.1155	45.1060	45.1294	45.1415	45.1002
27	45.1214	45.1010	45.1113	45.1139	45.1161

The overall effect of running these "pilot" GA runs in this manner was that a *systematic* and economical search of the GA parameter space could be thus achieved.

3.4 EXPERIMENTAL RESULTS AND THEIR INTERPRETATION

As noted above, Table 3.2 displays the "best" (i.e. minimum) objective function (σ) values obtained for each replicated GA run with 4000 function evaluations. The average and the spread of the replicated function evaluations for each GA run are graphically shown in Figure 3.2.

It is very clear that changing the parameter settings has a profound impact on the minimum, average and maximum "best" response value obtained in the replicated runs of the GA, each using 4000 iterations. The DOE approach allows us to also determine the contribution of each of the factors in the GA's convergence, and any interaction of their effects.

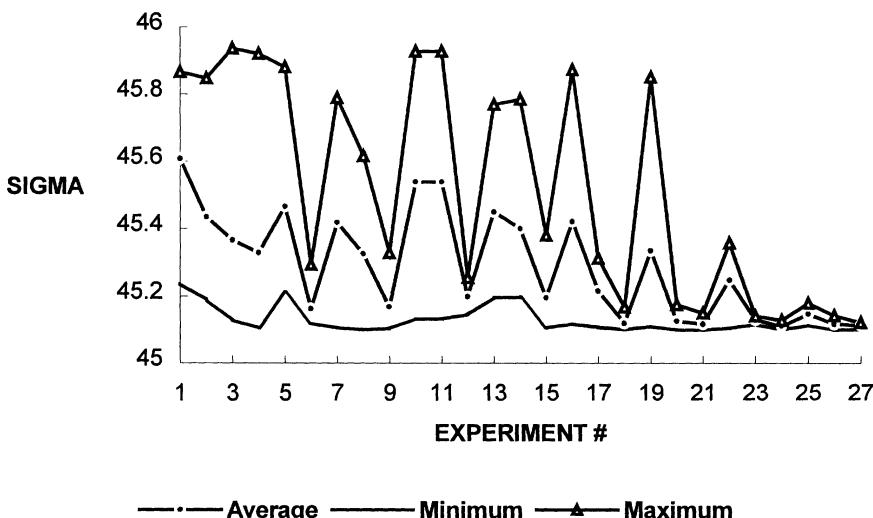


FIG. 3.2 CONVERGENCE TO "BEST" σ AS FUNCTION OF EXPERIMENTAL SETTING

A question may be asked: Are the factor effects (and perhaps their interaction) really significant? The answer may be found by subjecting the data in Table 3.2 to a statistical analysis of variance or ANOVA test (Montgomery, 1996). Table 3.3 shows the results of ANOVA, indicating that the effects of parameters p_s , p_c and p_m are each significant whereas in this particular problem instance, their interactions are not. This suggests that in order to optimize (maximize the convergence of GA in seeking the minimum of the response (here σ), we may consider adjusting the settings of p_s , p_c and p_m one factor at a time. We caution that this may not be the case always.

TABLE 3.3 ANALYSIS OF VARIANCE TESTS OF SIGNIFICANCE FOR $\text{MINIMUM } \sigma$ USING SUMS OF SQUARES

Source of Variation	SS	DF	MS	F_{calc}	Prob[$F > F_{\text{calc}}$]
WITHIN+RESIDUAL	5.01	108	0.05		
p_c	0.42	2	0.21	4.53	0.013
p_m	1.06	2	0.53	11.46	0.000
p_s	1.12	2	0.56	12.06	0.000
$p_c \times p_m$	0.08	4	0.02	0.43	0.787
$p_c \times p_s$	0.09	4	0.02	0.49	0.744
$p_m \times p_s$	0.20	4	0.05	1.08	0.369
$p_c \times p_m \times p_s$	0.21	8	0.03	0.56	0.805

Recall that the objective in the statistical experiments was to identify the parameter values (p_s , p_c , and p_m) that *together* would lead to an objective function value for σ (given by (3.4)) close to its true optimum (= 45.097, Castillo and Montgomery, 1993), while taking the smallest number of function evaluations. We proceeded as follows. (The material below is illustrative.)

First, we identified the different p_s , p_c , and p_m combinations that delivered objective function values (averaged over five randomized GA run replicates) close to the true optimum. Note that we also desired that the minimum objective function values obtained in the replicated GA runs with different random seeds for the "best" (p_s , p_c , and p_m) combination be *close to each other* or be "consistently good." If these conditions were both met by some combination of parameters

p_s , p_c , and p_m , that particular combination would be expected to speed up optimization *regardless of* the algorithm's starting condition. If several (p_s, p_c, p_m) sets would satisfy the consistency requirement, we would attempt to identify the specific set that would require *fewest* function evaluations to converge to within a stated tolerance of the true optimum solution. These parameter values would be regarded as the optimum set, expected to deliver both good off-line and good on-line performance.

The empirical identification of the optimum settings for these parameters began with the following specific observations:

- (i) For small population sizes (here $p_s = 0.7l$, which corresponds to Experiments 1 to 9 of the inner array in Table 3.4), the smallest objective function values (averaged over five replicates) are noted for the following two (p_c, p_m) combinations:

$$(p_c = 0.75, p_m = 3/l) \text{ and } (p_c = 0.95, p_m = 3/l)$$

These two combinations correspond to Experiments 6 and 9 respectively. Further, an examination of Figure 3.2 would suggest that the spreads of the minimum objective function values (over five replicates) corresponding to the above two (p_c, p_m) combinations are also smaller here than the spread for the other seven inner array runs.

A subsidiary objective in this study was to determine how fast the GA (parameterized in some manner) would approach the true optimum. Parameter combination $(p_c = 0.75, p_m = 3/l)$ required the smallest number of function evaluations to achieve an objective function value within a (pre-set) 1.86% tolerance of the true optimum. Furthermore, the spread of the number of function evaluations corresponding to $(p_c = 0.75, p_m = 3/l)$ (Experiment #6) was smaller than that for $(p_c = 0.95, p_m = 3/l)$ (Experiment #9).

Thus, we inferred that $(p_c = 0.75, p_m = 3/l)$ and $(p_c = 0.95, p_m = 3/l)$ would both give objective function values closer to the optimum. However, the combination $(p_c = 0.75, p_m = 3/l)$ took fewer function evaluations to do so. Therefore, we would conclude that for $p_s = 0.7l$

one should use $p_c = 0.75$ and $p_m = 3/l$, where l is the string length chosen (from precision considerations) to run the GA.

(ii) For "medium size" populations (population size $p_s = 1.3l$) we observed that minimum objective function value was reached with $p_c = 0.95$ and $p_m = 3/l$. This corresponds to Experiment #18 of Table 3.1. An examination of Figure 3.2 shows that this (p_c , p_m) combination had also the smallest spread (i.e., had greater consistency in delivering a solution closer to the true optimum). It was noted that for $p_s = 1.3l$, the combination $p_c = 0.95$, and $p_m = 3/l$ took fewer evaluations to converge to within the 1.86% tolerance of the true optimum. Also, with this setting the spread in the number of function evaluations was smallest when compared to the other experiments (between #10 and #18).

Therefore, we inferred that for medium population sizes, i.e., $p_s = 1.3l$, one should use $p_c = 0.95$ and $p_m = 3/l$.

(iii) For larger population sizes (e.g., with $p_s = 2.7l$) the (p_c , p_m) combination that produced small objective function values (with the smallest spread over 5 replicates) and also used the smallest number of function evaluations was ($p_c = 0.5$, $p_m = 1/l$).

Another significant observation here was that increasing the population size from a modest value ($0.7 \times$ string length) to a relatively high value ($\sim 3 \times$ string length), with the other two GA parameters (p_c and p_m) held at their "optimal" levels, did not noticeably reduce the number of function evaluations needed to achieve an objective function value in close proximity to its true optimum. However, when p_s was increased, the *spread* of optimum objective function values measured over five replicates generally diminished, improving consistency.

A close examination of the *minimum* objective function values reached in each successive generation suggests that the minimum value fluctuates considerably if the mutation probability (p_m) is "too high"—causing *more than* one mutation per chromosome. Here the algorithm appears to fail to converge to a particular objective function value. (By "convergence" we simply mean here that the changes in the

minimum value produced by the algorithm should become insignificant when it reaches within a close proximity of the global optima.) This result is consistent with the observed low mutation rates of well-adapted living organisms.

Empirical studies by us suggest that with high mutation probability (p_m) it may be difficult to "travel on a surface" defined by an equality constraint—when penalty functions are used to attain a target value. Further and extensive empirical studies appear necessary here. However, when we tested several different (p_c , p_m) combinations on problems using penalty functions (to ensure that the final solution would have its response in the close proximity of a stated target value), the results suggested that the crossover probability p_c should be kept in the 0.6-0.8 range (close to values prescribed by many GA users), and the expected number of bits mutated per chromosome should be kept less than 1.

In a large assortment of other optimization problems including complex production scheduling problems (see, for instance, Bagchi and Sriskandarajah, 1996) the parameter values identified by DOE rapidly produced final responses very close to the true optimum. Most of these experiments were used population sizes (p_s) of the order of l , the string length.

3.5 CHAPTER SUMMARY

Optimum parameterization of the GA has been long recognized as a basic necessity in the effective application of this novel meta-heuristic. Many attempts have been made in the past fifteen years to optimize not only p_s , p_c , and p_m , but also the number of crossover points, coding schemes, the percentage of population to be replaced in each generation, and the selection strategy. A central issue in this task is the interacting effect of the different parameters, which makes optimization a considerable challenge.

The parameterization method described in this chapter used pilot GA runs (of relatively short length) of the optimization problem at hand. Each pilot run was conducted within a *3³ factorial experimental design* (DOE) framework. The "response" observed in these experiments was

the variability noted in replicated GA results. This variability in turn was indicated by the distance between the Maximum and Minimum lines in Figure 3.2 as obtained by multiple starts of the GA.

Based on its performance on many numerical and combinatorial GA applications attempted, the DOE approach appears to be a valuable aid to guide the selection of GA parameters. DOE almost always leads to a better choice of parameters when compared with ad hoc specification of p_s , p_c , and p_m . In almost every case it actualizes superior on-line as well as off-line performance of the GA.

The idea of dynamically altering (or *adapting*) parameter values with appropriate "reward schemes" as proposed by Davis (1991) may be worth experimenting with. For instance, the probability of mutation (p_m) may be increased when the GA seems to "flatten out." Indeed, there is little reason to suggest that as the GA search evolves generation by generation, the initial parameter values will continue to be optimal. On the other hand, if one sets out to minimize some measure of deviations from the optimal evolution trajectory (such as the root mean square of errors), a robust tracking path for the parameters may be discovered—empirically.

Clearly, p_s , p_c , and p_m values obtained from empirical studies using some *specific* test problem cannot be claimed to be "broadly usable" or robust. Actually, there appears to be much interaction between the "optimum" parameter values and the function being optimized. In fact, we doubt if universally valid or "robust" parameter settings can ever be established for GAs. Therefore, a pilot study using short GA runs within the factorial design framework *and* the real problem itself as the test bed should prove well worth the extra effort. This appears to be a more productive pursuit than the use of the few isolated rules of thumb and parameterization guidelines that seem to exist (Bagchi, Templeton and Raman, 1995).

Above all, since parameter effects are noted to interact with the peculiarities of the solution space (the *fitness landscape*, Section 2.1) and often with each other, *in general* GA parameters should not be optimized by studying their effects *one-parameter-at-a-time*.

4

FLOWSHOP SCHEDULING

A *flowshop*, as described in Chapter 1, is characterized by unidirectional flow of work with a variety of jobs being processed sequentially in a one-pass manner. A *job shop*, on the other hand, involves processing on several machines without any "series" structure. In the past four decades extensive research has been done on both flowshop and job shop problems. This chapter evolves a synthesis of the best-known heuristic and meta-heuristic scheduling methods for single objective flowshop problems. We show that a strategy combining the strengths of the different methods produces solutions of good quality—*faster* than any single solution strategy.

This chapter summarizes the key methods, both algorithmic and heuristic, presently available for the flowshop. Note, however, that the majority of these methods solve only the *single objective* flow shop problem.

4.1 THE FLOWSHOP

In many manufacturing and assembly facilities a number of operations need to be done on every job. A "job" is thus a collection of operations to be performed on an item or unit with applicable technological constraints. Often, the operations must be done on all jobs in the same order. This implies that all jobs have to follow the same route, even if the jobs aren't identical. The machines are assumed to be set up in a series and such a processing environment is referred to as a flowshop (Baker, 1974).

Shop schedules are generally evaluated by aggregate quantities that involve information about all jobs, resulting in one-dimensional *performance measures*, usually expressed as a function of the set of job completion or processing times. A performance measure frequently used to compare the "quality" of two schedules is makespan, the total

time required to process all the jobs. Other common measures include mean flow time and job tardiness. Although the different performance measures are often correlated, one can construct an example for which a schedule may be good at one measure, but perform poorly on others.

The storage or buffer capacities in between successive machines in a flowshop may, at times, be virtually unlimited for all practical purposes. This is often the case when the products processed are physically small, e.g. printed circuit boards or small assemblies. It is then easy to store large quantities of such products between two machines and thus "uncouple" the machines.

A flowshop may be modeled as multiple processors working in series with jobs being multistage in nature. This means that it is essential to distinguish between a job and each of the processing operations that is a part of it. In a flowshop, a job (marked by " i ") acquires a particular notion, which is a collection of operations i_1, i_2, i_3 , etc. in which a special *precedence structure* applies. In particular, each operation after the first has exactly one direct predecessor and each operation before the last one has exactly one direct successor as shown in Figure 4.1

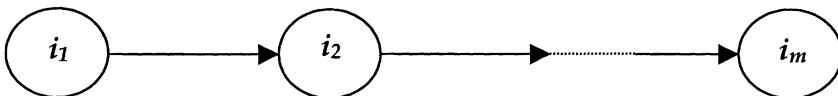


FIGURE 4.1 WORK FLOW IN FLOWSHOP

Thus each job requires a specific *immutable sequence* of operations to be carried out for it to be complete. This type of structure is sometimes referred as a linear precedence structure (Baker, 1974). Further, once started, an operation on a machine cannot be pre-empted.

The following notation is generally used in scheduling literature to describe a flowshop. The flowshop contains m different machines, and each job consists of m operations, which require a different machine each. Flow of work is unidirectional, from machine 1 to machine m . Put another way, a flowshop contains a natural machine order and it is possible to number the machines so that if the j^{th} operation of any job precedes its k^{th} operation, then the machine required by the j^{th} operation has a lower "number" than the machine required by the k^{th} operation. Thus machines in a flowshop are tagged $1, 2, \dots, m$ and the m operations of job i are correspondingly tagged $(i, 1), (i, 2), \dots, (i, m)$.

$1), (i, 2), \dots, (i, m)$. Note that each job can be treated as if it had exactly m operations. (If a certain job j requires no work on machine k , its processing time on machine k , i.e. t_{jk} would be 0.)

Since the arrangement of the processing stages (that we call *machines*) is determined by technological considerations (e.g., Clean parts → Rinse → Pickle to remove rust → Rinse → Deposit phosphate → ...), flowshop scheduling problems are really job sequencing problems. The challenge for the production planner in a flowshop is to find the optimum sequence in which the different jobs must proceed through the machines so as to meet one or more shop performance or service objectives (such as minimum time to finish all the jobs). From the point of view of the information available to the shop planner, flowshop-sequencing problems may be classified into following three categories:

- Deterministic flowshop problems.
- Stochastic flowshop problems.
- Fuzzy flowshop problems.

Deterministic problems presume that processing times of jobs are fixed (deterministic) and exactly known to the planner. In the stochastic case, processing times vary according to some given probability distributions. In the fuzzy decision context, a fuzzy (linguistically expressed) due date is assigned to each job to represent the degree of satisfaction of the planner for the completion of the job. Further, if any given job has a processing time equal to zero on one or more machines, the flowshop is called a *general flowshop problem*. Otherwise it is a *pure flowshop problem*.

The bulk of the research effort given to solve such problems during the past forty years has gone to the pure deterministic flowshop-sequencing problem involving only a single objective function. Still, no 'easy' algorithm that provides an optimal solution to the flowshop exists even now, except for the 2-machine problem and 3-machine problem under certain conditions. For the 2-machine problem Johnson (1954) developed a scheduling procedure that gives the minimum makespan for the jobs in a flowshop situation. By far perhaps the best method to solve such problems published to date is due to Ho and Chang (1991). The Ho-Chang heuristic is able to sequence the flowshop by taking makespan, mean flow time of jobs and mean utilization of the machines as three separate and distinct

objectives. Thus, the Ho-Chang heuristic is able to solve each of these three single-objective flowshop problems.

The three performance objectives—makespan, mean flow time of jobs and mean utilization of the machines—are certainly correlated, yet one can easily construct an example that shows that a schedule may be good at one measure but perform poorly on others. This raises the issue of multiobjective scheduling. However, with the exception of a handful of results, there has been relatively little research on flowshops with multiple planning or scheduling objectives. Rajendran (1995) has addressed flowshop scheduling with the triple objective of minimizing makespan, total flow time and machine idle time. His approach moves in steps. He first provides a heuristic to directionally minimize makespan and total flow time. Then he uses a heuristic preference relation as the basis for his first heuristic so that only potential job interchanges are checked for possible improvement with respect to these two objectives.

From a theoretical standpoint, even the single objective the flowshop problem is *NP*-complete and discussed widely in the literature (Garey, Johnson and Sethi, 1976; Lenstra, 1977). In the following subsections the commonly used algorithms for solving the pure deterministic flowshop-sequencing problem are reviewed.

4.2 FLOWSHOP MODEL FORMULATION

The common assumptions used in modeling the flowshop problem are as follows. The shop consists of m machines and there are n different jobs to be optimally sequenced through these machines. The following conditions apply in a flowshop:

- All n jobs are available for processing at time zero and each job follows identical routing through the machines.
- Unlimited storage exists between the machines. Each job requires m operations and each operation requires a different machine.
- Every machine processes only one job at one time and every job is processed on one machine at one time.
- Setup times for the operations are sequence-independent and are included in processing times.
- The machines are continuously available.
- Individual operations cannot be pre-empted.

Further,

1. Each of the n different jobs to be sequenced through the machines is an entity.
2. An operation, once started on a machine, must be completed before another operation is started on that machine.
3. Each job has m distinct operations, one to be performed on each machine.
4. Each job must be processed to completion.
5. Processing times (t_{ij}) are independent of the schedule (here the sequence of processing).
6. In-process inventory is allowed when necessary.
7. There is only one machine of each type in the shop.
8. Machines may be occasionally idle.
9. Machines never break down and are available throughout the scheduling period.
10. The technological (processing) constraints are known in advance and are immutable.
11. There is no randomness. In particular
 - (a) The number of jobs is known and fixed.
 - (b) The number of machines is known and fixed.
 - (c) The processing times are known and fixed, and
 - (d) All other quantities needed to define a particular problem are known and fixed.

The single machine flowshop will have the same makespan (the sum of the processing times for all the n jobs, no matter in what sequence the jobs are fed. However, if due dates are imposed, the mean tardiness and other performance measures of the shop may vary. There are $n!$ solutions (job sequences) possible here. A more interesting situation develops when the shop has more machines and since the jobs would have arbitrary processing times on each machine, idle times may develop, altering even the makespan from sequence to sequence. Johnson gave the first significant result for the two-machine flowshop in 1954 that minimizes makespan. This and some other significant methods are shown below.

4.3 THE TWO-MACHINE FLOWSHOP

The two-machine flowshop problem with makespan minimization as the objective is known as *Johnson's problem*. An optimal job sequence

for the two-machine problem can be determined by the celebrated Johnson's rule (Johnson, 1954). The ideas in Johnson's rule are of fundamental nature. They have provided a foundation for several good heuristics that were developed later for the m -machine flowshop problem as well.

Johnson's rule may be stated as follows. Suppose that there are n jobs with required processing times t_{i1} , $i = 1, 2, \dots, n$ on machine 1 and t_{i2} , $i = 1, 2, \dots, n$ on machine 2. An optimal sequence may then be characterized by the following rule for ordering pairs of jobs:

Theorem: Job i precedes job j in an optimal sequence if

$$\min [t_{i1}, t_{j2}] \leq \min [t_{i2}, t_{j1}]$$

This theorem constructs the job sequence that minimizes the makespan for a two-machine problem. The rule "builds" the sequence from the front and back toward the middle. It puts jobs with shortest times on machine 1 early in the sequence so that machine 2 can be brought into production quickly. Once machine 1 has finished processing all jobs, the jobs waiting processing at machine 2 have the smallest machine 2 times, so completion time for these jobs is minimized. Algorithmically expressed, the steps are as follows.

For the jobs yet to be sequenced, determine the minimum processing time of all $A_i = t_{i1}$ (processing time of i th job on machine 1) and $B_i = t_{i2}$ (processing time of i th job on machine 2).

1. If the minimum is associated with A_i , place the corresponding job in the earliest possible unscheduled position in the sequence achieved so far. If the minimum is associated with B_i , place the corresponding job in the latest unscheduled position in the sequence achieved so far.
2. Mark the job as being sequenced and scratch the associated A_i and B_i values.
3. If all the jobs are placed in the sequence go to the step 5, else go to step 1.

The sequence achieved is the optimal sequence.

Johnson's rule can be extended to the 3-machine flowshop problem, provided a condition is met—the second machine is not a bottleneck machine. For a machine not to be a bottleneck, it should not delay any

job. In other words, when a job is released from the first machine, it should immediately be processed on the second machine. If such is the case, by defining two factors A_i and B_i for each job i ($A_i = t_{i1} + t_{i2}$ and $B_i = t_{i2} + t_{i3}$) Johnson's rule for the 2-machine problem can be applied, treating factor A_i as the processing time on machine "1" and B_i as the time on machine "2."

When the production process has three or more machines, a general algorithm for finding the makespan-minimizing job sequence does not exist, so heuristics are normally used. However, as pointed out above, if the processing times for a three-machine problem meet a special condition, the three machine problem can be simplified to a two machine problem and solved using Johnson's rule (Pinedo, 1995).

4.4 SEQUENCING THE GENERAL m -MACHINE FLOWSHOP

In theory, integer programming and the branch-and-bound technique can be used to solve the flowshop problem optimally (Ignall and Schrage, 1965). However, these methods are not viable on large problems. Most scheduling problems including flowshop problems belong to the *NP-hard* class (Lenstra et al., 1977; French, 1982) for which the computational effort increases exponentially with problem size. To remedy this, researchers have continually focused on developing heuristic and other methods. Heuristic methods typically do not guarantee optimality of the final solution, but a final solution is reached quickly and is acceptable for practical use.

Several good heuristic methods have appeared in the past three decades to help minimize makespans for flowshops. A key distinction between these different heuristics is as follows. Some simply generate solutions (we call these *solution-generating heuristics*) while others attempt to improve existing solutions (the *solution-improvement heuristics*). Recent advances in *meta-heuristic search* methods that help conduct directed "intelligent" search of the solution space have brought yet new possibilities to rapidly find good schedules, even if they are not optimal. Some of these methods are context-independent and can be applied even when very little is known about the (mathematical) structure of the response functions being optimized. These methods are stochastic and now include genetic algorithms (GAs), tabu search, threshold acceptance and simulated annealing.

**TABLE 4.1 PROCESSING TIMES AND DUE DATES FOR A
49-JOB 15 m/c FLOWSHOP**

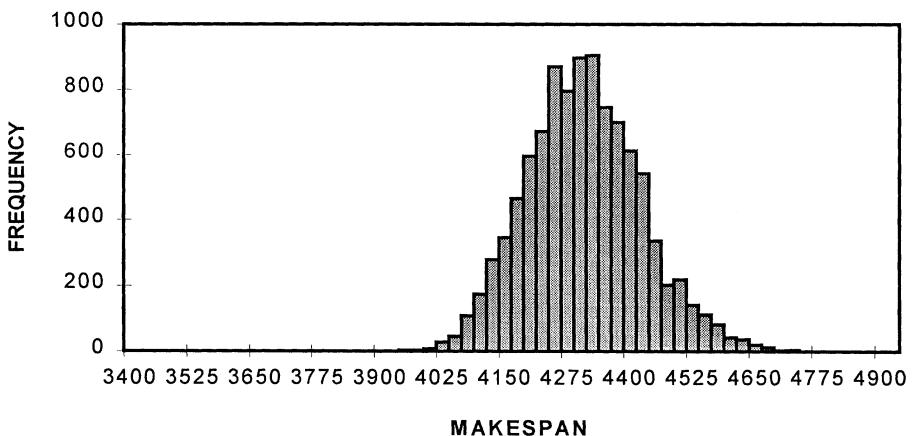
	m/c 1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	Due Date
Job 1	74	72	54	57	52	60	4	8	40	8	85	45	74	67	48	80
2	99	77	58	50	31	67	19	96	93	29	27	6	85	22	48	160
3	15	10	85	2	92	53	60	63	11	94	44	71	19	99	94	240
4	63	18	44	30	80	94	63	28	50	55	78	83	8	68	65	320
5	94	42	20	92	73	62	45	86	76	11	80	53	29	3	70	400
6	16	15	97	30	7	31	82	10	28	13	63	55	24	26	49	480
7	51	57	19	87	81	17	8	27	93	72	1	19	30	80	86	560
8	18	46	17	12	54	54	90	52	69	82	47	96	90	14	12	640
9	77	29	52	40	83	53	44	49	87	60	2	88	26	18	30	720
10	19	95	57	94	93	19	36	14	82	74	94	7	90	40	39	800
11	25	28	72	89	5	87	15	87	14	20	24	91	93	41	36	880
12	92	98	56	35	64	15	95	22	67	61	12	98	73	6	10	960
13	47	80	88	77	77	60	63	66	8	10	63	74	90	1	56	1040
14	38	84	99	21	13	73	0	26	68	99	9	72	42	43	27	1120
15	77	65	38	88	95	9	13	13	42	55	51	36	27	78	12	1200
16	22	81	30	45	63	94	44	78	98	57	26	85	61	82	71	1280
17	6	3	43	96	51	39	79	28	50	27	49	30	73	53	85	1360
18	32	28	61	62	18	76	92	50	5	26	1	53	33	79	17	1440
19	28	97	87	10	79	35	55	72	98	32	2	58	87	86	12	1520
20	16	73	45	90	76	86	75	38	58	49	61	90	23	92	24	1600
21	47	8	30	52	53	96	26	76	18	65	32	18	35	71	36	1680
22	65	90	73	28	34	58	20	10	46	94	38	34	0	17	72	1760
23	6	62	42	42	95	26	90	83	82	54	47	44	80	22	2	1840
24	39	12	28	15	77	22	46	17	19	18	83	5	28	55	96	1920
25	4	75	57	1	24	46	99	43	17	22	36	22	12	22	38	2000
26	14	97	19	81	92	44	66	92	71	66	43	61	11	42	82	2080
27	38	55	10	76	22	61	40	59	46	39	33	2	92	81	99	2160
28	82	38	7	12	2	28	88	89	2	93	88	60	60	53	82	2240
29	68	55	77	51	34	25	35	45	38	77	96	3	27	23	43	2320
30	86	4	95	81	63	51	73	80	1	10	11	47	87	80	54	2400
31	56	32	84	84	42	28	52	26	36	69	67	100	61	17	5	2480
32	68	27	31	8	86	85	39	54	78	85	80	48	2	91	91	2560
33	0	81	53	31	94	18	91	76	94	39	37	53	72	2	1	2640
34	19	42	68	86	34	81	12	90	47	63	11	71	80	15	99	2720
35	62	45	75	38	66	79	2	82	29	40	59	32	10	58	56	2800
36	16	23	73	23	66	80	37	62	71	50	18	6	1	5	71	2880
37	70	57	65	15	41	11	94	99	75	76	74	26	5	42	48	2960
38	80	2	95	25	47	96	10	9	22	12	60	50	40	38	42	3040
39	93	69	91	35	21	33	26	14	39	31	64	2	91	19	84	3120
40	89	86	17	55	24	64	38	49	62	36	76	14	31	60	78	3200
41	5	97	51	19	40	15	46	23	4	86	54	57	30	78	81	3280
42	86	98	63	24	26	21	41	1	53	51	88	93	84	60	20	3360
43	36	84	74	90	87	79	33	71	29	97	88	12	4	66	88	3440
44	43	77	40	96	18	75	9	89	13	96	17	31	20	80	37	3520
45	62	82	61	45	39	94	42	27	21	50	97	65	26	3	46	3600
46	58	49	11	78	82	89	45	84	71	53	39	48	31	44	100	3680
47	49	58	70	42	81	69	53	67	11	53	43	16	81	61	8	3760
48	94	85	5	41	90	33	21	55	71	77	45	95	99	55	32	3840
49	0	77	72	40	0	64	38	34	76	79	39	1	64	23	17	3920

Table 4.1 (Jayaram, 1998) displays the processing times of a typical flowshop problem in which 49 jobs are to be optimally sequenced through 15 machines to minimize makespan. $49!$ different solutions are possible here from among which the optimal solution must be

found. To illustrate the different methodologies developed, compared and contrasted in this chapter, we shall use this problem throughout as the test vehicle. For a problem with a search space as large as this one, one would first establish the opportunity—the *range* of the solutions possible. In order to do this, 10,000 different job sequences were randomly generated and their makespan evaluated.

Figure 4.1 presents the histogram of these makespans. The "best" solution contained in this sample had the makespan of 3928 while the solution with the largest makespan was 4750. This indicated the presence of a considerably wide search domain. In the subsequent sections we present the effect of applying various other job sequencing methods using reduction of makespan downward from 3900 as the objective.

FIGURE 4.2 DISTRIBUTION OF MAKESPANS OF 10,000 RANDOMLY GENERATED JOB SEQUENCES



4.5. HEURISTIC METHODS FOR FLOWSHOP SCHEDULING

To set the stage, in this section we recall the logic salient in the commonly used flowshop-sequencing heuristic methods. Also, to underscore the motivation for using GA, we highlight the relative effectiveness of these different methods.

4.5.2 Solution-Generating Heuristics

As noted, these methods simply generate solutions to the flowshop problem. Such methods may be subdivided into the following categories:

1. Those that apply Johnson's classic two-machine algorithm.
2. Those that generate and then use a "slope index" to qualify or prioritize job processing times to help align the jobs.
3. Those that attempt to minimize the total idle time of all machines.

Heuristics by Palmer (1965); Campbell, Dudek and Smith (1970); Gupta (1971); Dannenbring (1977); Nawaz, Enscore and Ham (1983); and Hundal and Rajgopal (1988) are among the *solution-generating heuristics* for the m -machine n -job flowshop problem with makespan minimization used as the scheduling criterion. Table 2 provides an overview of these different heuristics. The key features of these heuristics are subsequently recalled.

TABLE 4.2 BASIC RATIONALE FOR JOB SEQUENCING HEURISTICS

Heuristic	Johnson's Algorithm	Slope Index	Priority to High Total Processing Time Jobs	Total Machine Idle Time
Palmer (1965)		#		
Gupta (1971)	#			
Dannenbring (RA, 1977)	#	#		
CDS (1970)	#			
NEH (1983)			#	
Minimize Idle Time				#

- **Palmer Heuristic**

Palmer (1965) proposed a slope order index to sequence the jobs based on their processing times. Priority is given to jobs whose processing times tend to increase from machine to machine, while jobs with processing times that tend to decrease from machine to machine receive lower priority. Ordering of the jobs is done using a "slope index" (denoted by S_i for job i), calculated as

$$S_i = \sum_{j=1}^m (2j - m - 1)t_{ij} \quad i = 1, 2, \dots, n$$

where t_{ij} is the processing time of the i^{th} job on the j^{th} machine. A permutation schedule is then constructed by sequencing the jobs in descending order of S_i ,

$$S_{i1} \geq S_{i2} \geq \dots \geq S_{in}$$

For the problem shown in Table 4.1, Palmer heuristic produces a single sequence with makespan 3896.

- **Gupta Heuristic**

Gupta (1971) argued that the sequencing problem is really a problem of *sorting* n items so as to minimize the makespan. He extended a sorting function for Johnson's two and three machine cases to an appropriate function for the general m -machine case. He proposed an alternative slope index S_i for the jobs defined as

$$S_i = \frac{e_i}{\min_{1 \leq k \leq m-1} (t_{i,k} + t_{i,k+1})}$$

where

$$e_i = \begin{cases} 1 & \text{if } t_{i1} < t_{im} \\ -1 & \text{otherwise} \end{cases}$$

In the above expressions t_{ik} is the processing time of i^{th} job on machine k . For the problem in Table 4.1, the makespan produced by Gupta heuristic is 4313.

- **CDS Heuristic**

Campbell et al. (1970) treated the m machine problem as $(m-1)$ two-machine sub-problems and then constructed $(m-1)$ schedules using Johnson's algorithm. Out of these $(m-1)$ schedules the best one is chosen. Schedule number l ($l = 1, 2, \dots, m-1$) is constructed by solving a two-machine problem using Johnson's rule where two pseudofactors A_{il} (i^{th} job, first machine, l^{th} schedule) and B_{il} (i^{th} job, second machine, l^{th} schedule) are generated as follows and subsequently used.

$$A_{il} = \sum_{j=1}^l P_{i,j} \quad B_{il} = \sum_{j=1}^l P_{i,m-j+1}$$

For the problem shown in Table 4.1, the CDS heuristic produces a single sequence with makespan 3914.

- **Dannenbring Heuristic (RA)**

Dannenbring (1977) developed a procedure called rapid access (RA), which attempts to combine the advantages of Palmer's slope index and the CDS method. Instead of solving $(m-1)$ artificial two-machine problems, it solves only one artificial problem using Johnson's rule, in which the processing times are determined from a weighting scheme as follows:

$$A_{il} = \sum_{j=1}^l w_{j1} P_{i,j} \quad B_{il} = \sum_{j=1}^l w_{j2} P_{i,m-j+1}$$

where two sets of weight factors $\{w_{j,1}\}$ and $\{w_{j,2}\}$ are defined as

$$W_1 = \{w_{j,1} \mid j = 1, 2, \dots, m\} = \{m, m-1, \dots, 2, 1\}$$

$$W_2 = \{w_{j,2} \mid j = 1, 2, \dots, m\} = \{1, 2, \dots, m, m-1\}$$

For the problem shown in Table 1, this heuristic produces a single sequence with makespan 3868.

- **NEH Heuristic**

The Nawaz et al. (1983) heuristic is based on the philosophy that a job with higher total processing time needs more attention than that of one with a lower total processing time. Because an exhaustive search technique is employed in this procedure, the enumeration effort in this method is large and its extent is quantified by $(n(n+1)/2) - 1$ where n equals to number of jobs to be sequenced. NEH builds the final schedule in a constructive way shown below, adding a new job at each step of iteration and finding the best partial schedule.

Algorithm

Step 1. Order the n jobs in descending order of total processing times on the machines.

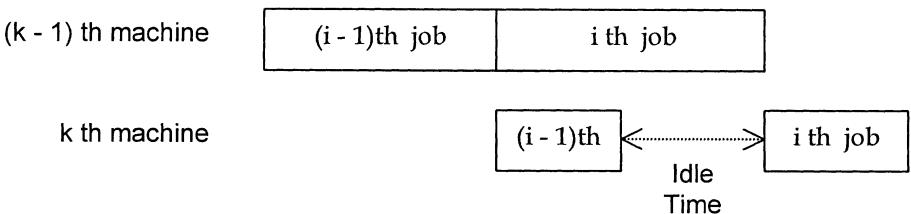
- Step 2.** Take the first two jobs and schedule them in order to minimize the partial makespan as if there were only these two jobs to be processed.
- Step 3.** For $k = 3$ to n do: insert the k^{th} job at the place, among the k possible ones, which minimizes the partial makespan.

For the problem shown in Table 4.1, NEH heuristic produces a single sequence with makespan 3574.

- **The "Minimize Idle Time" Heuristic**

The basic principle here is to schedule the jobs so as to minimize the total idle times of the machines. It can be shown that for a flowshop minimizing machine idle time is equivalent to minimizing total makespan. Therefore a heuristic was devised to build up a single chain sequence: the next job is always selected so as to minimize total between-job delay and that job is added to the end of the "chain" from the set of unscheduled jobs. The procedure is as follows.

- a. Find the sum of the processing times for each job. Arrange the jobs in the ascending order of their summed times. Schedule the job with minimum sum in the first position and calculate its completion time on each machine.
- b. Schedule each of the unscheduled (candidate) jobs next to the last confirmed scheduled job in the sequence developed so far and calculate the completion times of each such candidate job.
- c. Calculate the *total* machine idle time for each candidate job. Machine k will experience idle time if the completion time of i^{th} job on the $(k-1)^{\text{th}}$ machine is greater than the completion time of $(i-1)^{\text{th}}$ job on the k^{th} machine as shown below.



- d. The next job to be confirmed in the schedule will be the one with minimum total idle time.
- e. If all jobs are scheduled stop, else go to Step b.

The makespan for the sequence to the problem in Table 4.1 produced by the above heuristic is 4217.

4.5.2 Solution-Improvement Heuristics

An "improvement heuristic" takes an initial given job sequence and through modifications attempts to find a better solution. Some of the improvement heuristics are RACS (Dannenbring, 1977), RAES (Dannenbring, 1977) and Ho-Chang (Ho and Chang, 1991). Each uses makespan minimization as its primary objective.

- **The RACS (Rapid Access with Close Order Search) Heuristic**

Dannenbring (1977) observed that in many cases a simple transposition of adjacent jobs in the solution sequence obtained by RA heuristic described above yielded optimal solutions. For this reason he suggested a *one-stage* improvement process. "Neighbors" of the rapid access solution are defined as all new sequences that can be formed by the transposition of a single pair of adjacent jobs. Each of these ($n - 1$) neighbors is examined for a possible improvement in makespan.

- **The RAES (Rapid Access with Extensive Search) Heuristic**

Instead of terminating the search after one set of interchange, the rapid access with extensive search heuristic (Dannenbring, 1977) uses the best immediate neighbor to generate further neighbors. This process is continued as long as new solutions with improved makespans are found.

- **Ho-Chang Heuristic**

Ho and Chang (1991) proposed a different improvement heuristic that improves the incumbent sequence by the "gap" concept. This heuristic attempts to minimize the gap between successive operations performed on a job in solutions generated by other heuristics. The heuristic works as follows.

Step 1: Apply any heuristic method to get an initial solution, called the incumbent solution.

Step 2: Calculate the overall *revised* gaps, d_{ij}^R , as follows:

$$d_{ij}^R = \sum_{k=1}^{m-1} d_{ij}^k \delta_{ij}^k \quad i, j = 1, 2, \dots, n, i \neq j.$$

$k = 1, 2, \dots, m-1.$

Where

$$\delta_{ij}^k = \begin{cases} factor(k) & \text{if } d_{ij}^k < 0 \\ 1 & \text{otherwise} \end{cases}$$

$$factor(k) = \frac{(1.0 - 0.1)}{(m-2)}(m - k - 1) + 0.1 \quad k = 1, 2, \dots, m-1.$$

$$D_{ij}^k = t_{i,k+1} - t_{j,k} \quad \text{for } i, j = 1, 2, \dots, n, \quad i \neq j; \quad k = 1, 2, \dots, m-1.$$

Step 3: Let l ($l = 1, 2, \dots, n$) represent the positions in the incumbent solution and P_l represent the job at the position l . Set a to 1 and b to n .

Step 4: Search for the largest value (called X) of $d_{P_a P_l}^R$ where l is between a and b (both exclusive). Let u represent this l associated with X.

Step 5: Search for the smallest value (called Y) of $d_{P_b P_l}^R$ where l is between a and b . Let v represent this l associated with Y.

Step 6: If $(X < 0), (Y > 0)$ and $(|X| \leq |Y|)$, then go to Step 9.

Step 7: If $(X < 0), (Y > 0)$ and $(|X| \geq |Y|)$, then go to Step 10.

Step 8: If $(|X| > |Y|)$, then go to 9. Otherwise go to 10.

Step 9: Let $a = a + 1$, and swap the jobs in positions a and u . Go to Step 11.

Step 10: Let $b = b - 1$, and swap the jobs in positions b and v . Go to Step 11.

Step 11: If the new schedule is better than the incumbent in terms of a performance measure, it becomes the new incumbent solution. Otherwise, swap back to the original incumbent solution.

Step 12: If $b = a + 2$, then stop. Otherwise go to Step 4.

In the past decade, this particular heuristic has been compared in the literature to several other heuristics and it has been shown to be superior to many of the other available heuristic methods that attempt to minimize makespan in a flowshop.

Genetic algorithms have been already applied successfully to solve flowshop problems. The following description is a summary of the different approaches followed by contemporary researchers using GA to sequence flowshops. When applied to flowshop scheduling, GAs can view job sequences directly as "chromosomes" (the candidate schedules or solutions), that then constitute the members of a GA population. Subsequently, each individual (a schedule) is characterized (merited) by its fitness (e.g. by its makespan value). For a flowshop a chromosome would represent a job sequence on a machine, such as [1 3 2 4 5]. Fitness evaluation for a sequence would go, for instance, as the *smaller* its makespan, the "fitter" it is. As the GA executes, in each generation the "fittest" chromosomes are encouraged to reproduce while the least fit "die."

4.6 DARWINIAN AND LAMARCKIAN GENETIC ALGORITHMS

As we stated in Chapter 2, genetic algorithms (GAs) belong to the class of heuristic optimization techniques that utilize randomization as well as directed *smart* search to seek the global optima. We also saw in Chapter 2 how the creation of GAs was inspired by theories about natural evolutionary processes. Pinedo (1995) provides a clear rendering of GAs in the context of scheduling. Increasingly, GAs are being found to be more general and abstract (context independent) than the other popular heuristic techniques presently available. As a result, many practitioners have already turned to GAs to solve the more difficult and large sequencing, lot sizing, and even classroom scheduling problems (Carter, 1997).

As mentioned in Chapter 2, GAs discover solutions to global optimization problems *adaptively*, looking for small, local improvements rather than big jumps in solution quality. Also, while most stochastic search methods operate on a single solution to the problem at hand, GAs operate on a *population* of solutions. To use GA,

however, one must first *encode* the solutions to the problem in a chromosome-like *structure* (Goldberg, 1989). The procedure then applies *crossover* and *mutation* and other processes inspired by natural evolution to the individuals in the population to generate new individuals (solutions). The GA uses various selection criteria to pick the best individuals for *mating* so as to produce superior solutions by combining parts of parent solutions akin to genetically breeding racehorses or superior strains of food crop. The objective function of the problem being solved determines how "good" each individual is.

Sketched below are the schemas of two genetic algorithms. The first one is essentially the construct of Holland (1972, 1975, 1992), based on Darwin's theory of evolution. Literature refers to this particular GA as the Darwinian GA or S(imple) GA or SGA. The second GA is based on Lamarck's theory of evolution.

Darwinian GA (also called SGA)

```

begin
  t ← 0;
  initialize P(t) ;
  evaluate P(t) ;
  while (not termination condition) do
    recombine P(t) by crossover & mutation to give C(t) ;
      evaluate C(t) ;
      select P(t + 1) from P(t) and C(t) ;
      t ← t + 1;
  end;
end;
```

Lamarckian GA

```

begin
  t ← 0;
  initialize P(t) ;
  evaluate P(t) ;
  while (not termination condition) do
    recombine P(t) by crossover & mutation to give C(t) ;
      locally climb C(t) by learning;
      evaluate C(t) ;
      select P(t + 1) from P(t) and C(t) ;
      t ← t + 1;
  end;
end;
```

The Darwinian GA is the closest emulation of natural evolution, except that population size is kept constant here. In practice, several variations of crossover and mutation processes may be used. Selection is based on fitness.

Grefenstette (1991), Davidor (1991), and Kennedy (1993) suggest hybrid approaches to be used in the *local climb* step in Lamarckian GA. The intention is to inject some "smarts" into the offspring before returning it to evaluation. Hybridization of GA is conventionally attempted by (a) adapting the GA operators, or by (b) incorporating conventional local search heuristics in the "locally climb" step shown above (Gen and Cheng, 1997). With hybridization, the GA performs global exploration while the heuristics do local search around the chromosomes in the population. Because of complimentarity, the combined method outperforms either method operating alone.

4.6.1 THE GA BY GEN, TSUJIMURA AND KUBOTA

Solution Representation

Because the flowshop problem is essentially a permutation problem, Gen, Tsujimura and Kubota (1994) made the natural choice: they used the permutation of jobs as the representation scheme of chromosomes. Thus, for them the k th chromosome v_k represented by [3 2 4 1] implies that the job sequence is j_3, j_2, j_4, j_1 .

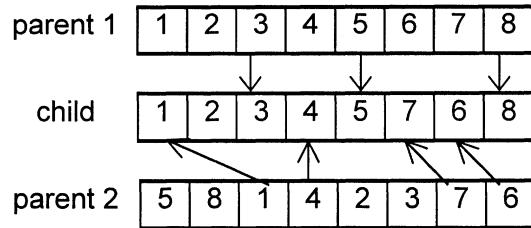
Evaluation Function

Gen et al. used a simple way to determine the fitness for each chromosome: they used the inverse of makespan for it. Thus, if c_{\max}^k denotes the makespan for v_k , then its fitness would be $1/c_{\max}^k$.

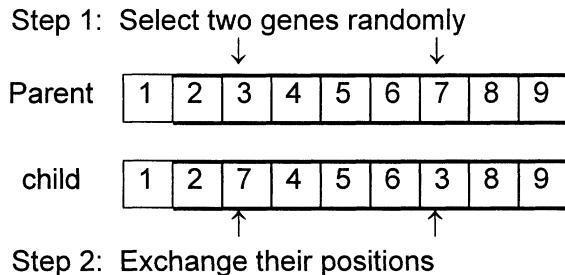
Crossover and Mutation

A *crossover* may combine some features of two parent chromosomes to create progenies inheriting some characteristics from each parent. A "repair" scheme may be set up to ensure that only feasible progeny sequences are produced. Thus the child [1 3 2 2 5] would require some "repair" (replacement of the repeated job (2) by the missing job (4)). Several different crossover schemes were used by Gen et al. These included the one-point crossover, two-point crossover, position-based crossover, PMX (partially mapped crossover), OX (order crossover), and CX (cycle crossover). An example of position-based

crossover that shows the transfer of jobs from two "parents" to form a feasible "child" would be



A *mutation* in a parent chromosome may be an adjacent pairwise interchange of jobs or some variation of it in the corresponding sequence. Mutation here is designed to perform random exchange of jobs ("genes") between two randomly selected positions on a target chromosome. An example is shown below.



4.6.2 THE GA BY REEVES

Reeves (1995) proposed a different implementation of the GA method for the flowshop. He tested his algorithm on Taillard's benchmark problems (Taillard, 1993) and concluded that simulated annealing (another meta-heuristic search method created by Metropolis et al. (1953)) and GA produce comparable results for flowshop sequencing for most sizes and types of problems, with GA performing somewhat better for larger problems.

According to Reeves, for large problems GA reaches near-optimal solutions faster. However, in an earlier such comparison done by Ogbu and Smith (1991) simulated annealing is shown to have an edge.

4.6.3 A HYBRID GA BY ISHIBUCHI, YAMAMOTO, MURATA AND TANAKA

Ishibuchi, Yamamoto, Murata and Tanaka (1994) created some fuzzy flowshop scheduling problems and applied GA and neighborhood search algorithms together in a piggyback manner to solve them. They show that for such problems GA hybridized neighborhood search performs well.

4.6.4 A GA BY CHEN, VEMPATI AND ALJABER

Chen, Vempati and Aljaber (1995) used makespan minimization as the criterion to tackle flowshop-sequencing. They used Goldberg's PMX (partially mapped crossover) operator. They note that a good initial population produced, for example, by the CDS method (Section 4.5) gives the GA considerable advantage.

The GA results Chen et al. produced are somewhat better than those obtained by the Ho-Chang heuristic (Section 4.5), however, they note that the initial population, population size, selection probability and the type of the GA operator used—all affect the final results.

4.6.5 A GA FOR THE CONTINUOUS FLOWSHOP

Chen, Neppalli and Aljaber (1996) have provided a GA-based approach to tackle the continuous flowshop problem in which the intermediate storage for partially finished jobs does not exist, as in a steel rolling mill. This problem is *NP-hard* and an earlier attempt has been made to solve it as a travelling salesman problem (Reddi and Ramamoorthy, 1972). Total flow time of the jobs is taken here as the performance criterion. Chen et al. represent the solution as a sequence of jobs. PMX is used as the crossover operator. A noteworthy feature of this work is that the investigators also attempted optimization of the GA parameters (p_c , p_m , population size, etc.) using a metalevel GA as suggested by Grefenstette (1986) (cf. the design of experiments-based method of Chapter 3). They note that the choice of these parameters has a large influence on the final result. Also, they state that a good starting population helps out very significantly.

4.6.6 TWO HYBRID GAs BY MURATA, ISHIBUCHI AND TANAKA

Murata, Ishibuchi and Tanaka (1996) present yet two other hybridized GA variations to sequence the flowshop. Makespan minimization is still the objective. The investigators used four distinct types of crossover operators including the common one-point, two-point and position-based variations, and for different mutation methods. Three variations of selection probability estimators were used. GA parameters were chosen here using an extensive, exhaustive search conducted by simulation with test problems.

Murata, Ishibuchi and Tanaka compared their results to local search methods, tabu search and simulated annealing. They conclude that Darwinian GAs perform much better than purely random search, but are a bit inferior to local search, tabu search and simulated annealing. However, hybridizing the GA by local search improves its performance a great deal.

4.6.7 COMBINING SIMULATED ANNEALING AND KNOWLEDGE

Zegrodi and Enkawa (1995) have attempted to minimize makespan in a flowshop by incorporating simulated annealing with problem-specific knowledge, provided in the form of "move desirability of jobs." The move desirability index for a job was determined using several rules obtained by culling the flowshop scheduling literature. The investigators show that such a combination can form a better solution strategy than any single heuristic or search method used alone.

4.7 FLOWSHOP SEQUENCING BY GA: AN ILLUSTRATION

As we indicated in Chapter 3, a critical difficulty in effectively using GAs is that the various GA parameters must be correctly chosen to ensure the GA's satisfactory on-line and off-line convergence. To this end, as described in Chapter 3, the GA may be parameterized using a *design-of-experiments* (DOE) approach employing pilot GA runs. In order to illustrate the method, we use again the 15-m/c 49-job flowshop problem of Table 4.1, the objective being minimization of

makespan. Table 4.3 shows the DOE matrix used here to determine the factor effects.

Figure 4.3 displays the factor effects discovered, using the simple (Darwinian) GA as the test bed and makespan minimization as the objective. The figure shows the relatively strong effects of population size (p_s) and probability of mutation (p_m) and relatively weak effect of the value of probability of crossover (p_c) in the ranges tested. The results indicate that a *high* p_s , and *low* p_m and p_c near 0.7 would be the best parameter combination to apply GA here in a production mode. Note that DOE separates out the effect of *each* parameter on the GA's convergence, a contrast with the blind and ad hoc use of parameter values or arbitrary "rules-of-thumb" parameterization guidelines. Note, however, that the DOE method is exploratory and the optimum parameter values found are likely to be *problem-specific*.

TABLE 4.3 DESIGNED EXPERIMENTS TO UNCOVER FACTOR EFFECTS ON THE DARWINIAN GA's PERFORMANCE

Expt. No.	Population Size	Pc	Pm	Avg Makespan	Max	Min
1	100	0.9	0.05	3701	3772	3645
2	100	0.9	0.005	3610	3684	3536
3	100	0.7	0.05	3687	3747	3624
4	100	0.7	0.005	3628	3694	3542
5	20	0.9	0.05	3710	3780	3633
6	20	0.9	0.005	3706	3809	3623
7	20	0.7	0.05	3699	3776	3587
8	20	0.7	0.005	3711	3802	3645

FIGURE 4.3 FACTOR EFFECTS ON THE DISCOVERY OF GOOD SEQUENCES BY GA

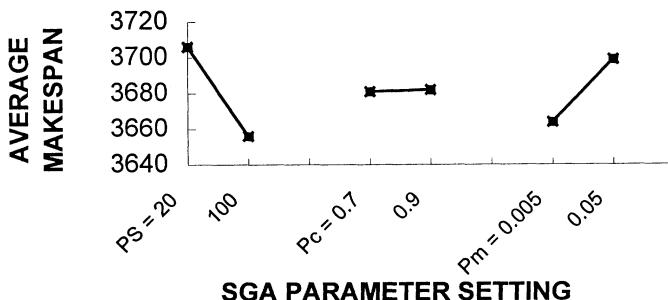


FIGURE 4.4 THE CHARACTERISTIC CONVERGENCE OF A TYPICAL GA

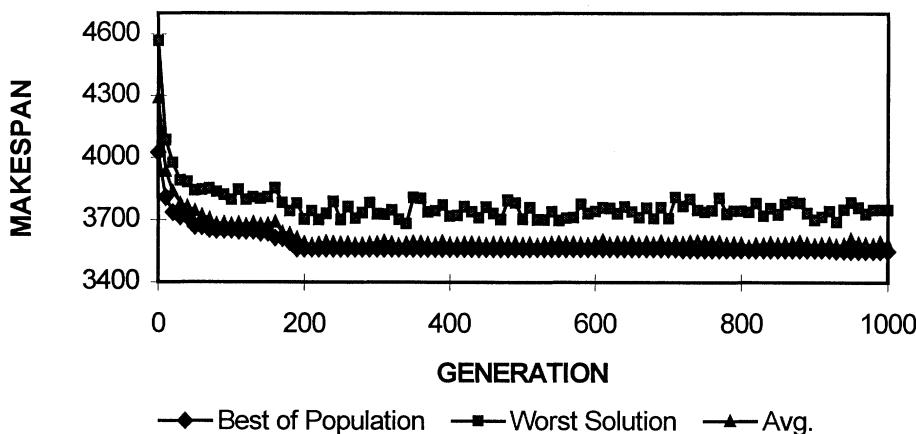


Figure 4.4 indicates the rate at which increasingly efficient sequences were found by one typical application of the Darwinian GA. A poorly parameterized GA may not converge at all! The average best makespan found after 1000 generations with $p_s = 100$, $p_c = 0.7$ and $p_m = 0.005$ and re-starting the GA with 24 different seeds was 3525. The comparative results tabulated below demonstrate the broad utility of using GA. On an average, a properly parameterized GA is seen to produce solutions of quality comparable to those found by the best heuristic methods available. Indeed, this is one key reason for the increasing popularity of GAs in solving combinatorial optimization problems.

Solution Method	Random Search	CDS Heuristic	NEH Heuristic	RA Heuristic	Palmer Heuristic	Darwinian GA avg.
Best Makespan	3928	3914	3574	3868	3896	3525

4.8 DARWINIAN AND LAMARCKIAN THEORIES OF NATURAL EVOLUTION

According to John Maynard Smith (Smith, 1993), the tenable idea today about how evolution occurred is what Darwin proposed: evolution happened through successive stages of adaptation followed by natural selection. One view put forth as an alternative to Darwin's proposal is the "Lamarckian" theory. This theory suggests that

evolution occurs when individual organisms adapt *during* their lifetimes (e.g., growing long necks due to continual stretching to reach tall trees or acquiring strong muscles by the continuous exercising of certain body parts). By this theory the "adapted" organisms then pass on those adaptations to their offspring.

Note that both Darwinian and Lamarckian theories moor on *adaptation*, the term adaptation (*ad + aptare*, to fit to) designating any process whereby a structure is progressively modified to give better performance in its environment. In general, adaptation may alter the organism's structure (its physical or genetic makeup), or its behavior. Note, however, that Darwin and Lamarck differed in how they each viewed adaptation playing a role in evolution.

Darwin's theory is affixed on a key process that he called *natural selection*. Darwin's view of adaptation accepts life as an active equilibrium between the living organism and its surrounding environment. This equilibrium can be maintained only if the environment suits the organism, which is then said to be "adapted" to that environment. Subsequently, natural selection moves into gear: the well-adapted individuals survive to procreate while the least fit die off. Thus, according to Darwin, natural selection, assisted by local adaptation, mechanizes evolution in nature. Local adaptation is aided by the presence of adaptive or advantageous genetic variations in organisms. Holland's GA is Darwinian.

The Lamarckian explanation of evolution has been called "inheritance of acquired characters." This viewpoint suggests that adaptation occurs *during* the lifetime of an organism and then such adaptation is passed on by organisms to their progeny. Thus, useful characteristics may be *acquired* during the organism's lifetime, which are then passed on naturally to the progeny. For instance, by this theory, the son of an ironsmith would inherit a strong right arm by virtue of being his father's biological progeny. Thus a lineage of "strong right arm" progeny would evolve. However, biologists and others have gathered very little evidence till now to support the "acquired characteristics" or Lamarckian theory of evolution. In fact, evidence appears to be overridingly to counter the Lamarckian theory of evolution and most biologists believe it to be no more a matter of controversy.

Smith (1993) points out the significant role that *learning* plays in an organism's development. Natural selection is a slow process and

evolution occurring by natural selection and genetic inheritance alone can get "stuck" at some stage of evolution for a long period, if sources of variation are relatively few. However, as Smith cites, the example of "historical evolution," a term reserved for the social, scientific and technological advances that mankind has enjoyed in the past 100 years is truly remarkable. No hereditary mechanism or process has been involved here whereas the degree of changes achieved is extraordinary.

Therefore, a *better* adaptive plan would possibly be one that would combine genetically inherited adaptation with learning, as one *combining* Darwinian adaptation (occurring through genetic variation and natural selection) with Lamarckian adaptation (transfer of acquired characteristics). Our study involving flowshops shows that in GA-type, artificially produced evolution such a scheme yields significant improvement in the rate of evolving superior individuals. Notwithstanding the limits of biological evolution (in which $\text{DNA} \rightarrow \text{RNA} \rightarrow \text{Protein}$ transformation is possible but not $\text{RNA} \leftarrow \text{Protein}$, see "The Central Dogma" in Smith, 1993 and Russell, 1998), a critical advantage we have in the artificial world of GAs is that in it we are often able to force acquired or "learned" characteristics back all the way to "alter the organism's genetic material" with no particular difficulty. The present research demonstrates that when this can be done, the results are remarkable when compared with the performance of the simple, "natural" or Darwinian GA.

A different search scheme may work as follows. Once a GA solution is improved (even if by some learning process), the changes that brought about the improvement can become an integral part of the solution: the revised chromosome acquires the structural changes, thereby replacing the genetic makeup of the original solution. (This is entirely possible with many chromosome-coding schemes used in shop scheduling or TSP.) The population now has an improved solution as if the original solution in it was put through a coaching camp. The key challenge in implementing such modifications to the basic GA would be to first somehow *improve* a given individual (a solution in GA) and then capture this improvement in the genetic makeup (of that individual). Once this is achieved, the new (superior) chromosome would participate in the mating process to pass on its superior genes to the progeny through normal GA processing.

Note that such a scheme would be different from the blind or the probabilistic application of hybrid methods to assist GA search. The subsequent portions of this chapter summarize the outcome of following this new scheme with flowshop sequencing used as the problem-solving context.

4.9 SOME INSPIRING RESULTS OF USING LAMARCKISM

Several possibilities spring for seeking such improvements in the basic GA methodology. These grow from some obvious questions such as the following. What is the effect of hybridizing the Darwinian GA with the various flowshop sequencing heuristic methods—CDS, NEH, RA, etc.? This question has already been explored in the literature (Gen and Cheng, 1997) and in a broad sense researchers have found hybridizing to be advantageous. What if we *start* GA search by having already applied the different solution-generating heuristics (Section 3 above) to the starting population? Would the advantage of starting at such "higher ground" be advantageous *in general*? What if only *a few* (even one) heuristic sequences are introduced in the starting population?

Subsequently, sharper questions may be asked. These would aim at maximizing the advantages, if any, of possibly synergizing analytical and heuristic methods, the Darwinian GA, Lamarckian GA, as well as any other innovation.

Some answers are immediately obvious. For instance, because the sequence-generating heuristics produce a unique sequence for a given flowshop problem, these methods cannot be used for a Lamarckian *local climb*: They would rearrange the sequence in accordance with the logic of CDS, NEH etc., *destroying* any higher ground already reached by GA. Thus, any hybridizing attempted must incorporate a *solution improvement* heuristic, such as RACS, RAES or Ho-Chang. Hence a promising prospect to be explored would be to construct a Lamarckian GA by suitably incorporating one or more solution-improvement heuristic(s) in it.

In this study a wide range of strategic combinations of Lamarckian solution methods incorporating the Ho-Chang improvement heuristic were tested on twenty five different randomly generated flowshop

problems. Each GA application was optimally parameterized by statistically designed experiments before the "production" run was made. These results are indicative of the overall comparative performance of the different strategies. Illustrative results for the 49-job 15-m/c problem considered earlier, iterated for 1000 GA generations, are shown in Table 4.4.

The salient findings of numerical experiments conducted on the different randomly generated flowshop problems were as follows.

- The Darwinian GA generally produced solutions superior to what could be produced by solution-generating heuristics such as Palmer, CDS, NEH or RA applied alone.
- However, for every *strategic* combination of solution methods tested (as indicated, for instance, by Table 4.4), the pure Darwinian GA operating alone and started with a randomly generated initial population led to statistically larger makespans (i.e., inferior job sequences).
- Whenever the initial population was given a "kick start" by filling it up with all-identical Palmer, CDS, NEH or the RA heuristic solutions, results improved statistically as established by the Wilcoxon signed rank test.
- When Lamarckism was introduced by improving the best solution by "compressing" it by invoking the Ho-Chang improvement heuristic, *whenever the best solution did not improve for several generations*, results improved further, statistically.
- Setting the probability of mutation (p_m) equal to zero got the Lamarckian procedure "stuck" as no variation beyond a point could occur only through crossover.
- In every case we produced solutions better than what was produced by random search or by the stand-alone application of the solution generating heuristic methods—Palmer, NEH, CDS and RA.

Similar performance improvement is reported in job shop scheduling by GA using the shifting bottleneck heuristic (Balas and Vazacopoulos, 1994) as the solution improvement heuristic.

TABLE 4.4 BEST MAKESPANS PRODUCED BY DIFFERENT RANDOM SEEDS FOR VARIOUS LAMARCKIAN GAS

Random Seed #	Starting Population = 100% Palmer	100% Palmer assisted by Ho-Chang	Starting Population = 100% NEH	100% NEH assisted by Ho-Chang	Starting Population = 100% CDS	100% CDS assisted by Ho-Chang	Starting Population = 100% RA	100% RA assisted by Ho-Chang	Pure Darwinian GA
1	3483	3467	3475	3461	3512	3453	3524	3468	3593
2	3455	3464	3489	3483	3469	3456	3496	3484	3513
3	3467	3441	3480	3466	3506	3493	3482	3476	3507
4	3485	3471	3482	3468	3474	3464	3514	3501	3528
5	3493	3452	3485	3496	3482	3460	3476	3464	3494
6	3470	3469	3457	3454	3518	3487	3458	3473	3539
7	3472	3456	3474	3487	3482	3479	3474	3459	3518
8	3462	3487	3516	3478	3483	3473	3457	3462	3567
9	3469	3461	3485	3477	3483	3492	3523	3465	3492
10	3488	3473	3474	3465	3509	3494	3522	3479	3488
11	3468	3441	3516	3455	3501	3474	3492	3468	3569
12	3481	3465	3480	3496	3481	3465	3495	3478	3557
13	3480	3459	3490	3454	3495	3481	3500	3484	3484
14	3454	3456	3481	3457	3498	3464	3474	3463	3510
15	3491	3463	3486	3500	3496	3503	3513	3491	3483
16	3468	3491	3487	3477	3470	3488	3454	3489	3524
17	3480	3466	3468	3477	3474	3465	3491	3455	3523
18	3523	3460	3501	3486	3493	3481	3501	3469	3504
19	3483	3475	3498	3476	3520	3484	3489	3504	3535
20	3504	3459	3454	3475	3480	3480	3497	3488	3525
21	3475	3447	3503	3492	3511	3481	3472	3498	3498
22	3470	3486	3452	3493	3500	3470	3473	3477	3560
23	3468	3453	3523	3491	3474	3492	3497	3482	3540
24	3472	3465	3492	3525	3494	3484	3483	3487	3542
Average	3478	3464	3485	3479	3492	3478	3490	3478	3525
Max-Min Span									

Figures 4.5 to 4.8 display the relative convergence of typical Darwinian and Lamarckian GA runs with the starting solutions being 100% of Palmer, NEH, CDS or RA solutions as noted on the figures.

In these instances Lamarckism was introduced by invoking the Ho-Chang heuristic to compress the best solution (job sequence) each time the best solution at hand did not improve by Darwinian GA processing for five successive generations.

FIGURE 4.5 BEST SOLUTIONS FOUND BY DARWINIAN, PALMER-INITIATED DARWINIAN AND PALMER-LAMARCKIAN GA

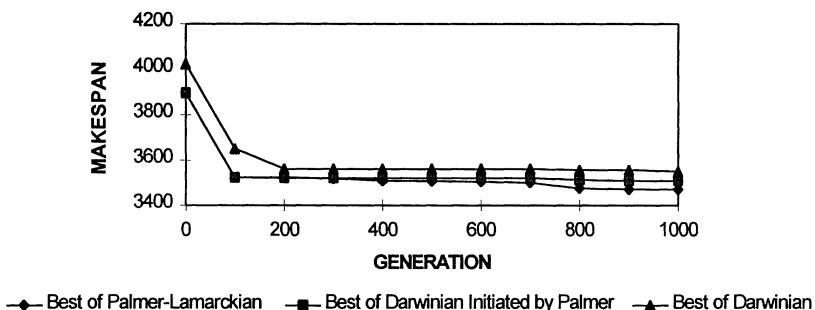


FIGURE 4.6 BEST SOLUTIONS FOUND BY DARWINIAN, NEH-INITIATED DARWINIAN AND NEH-LAMARCKIAN GA

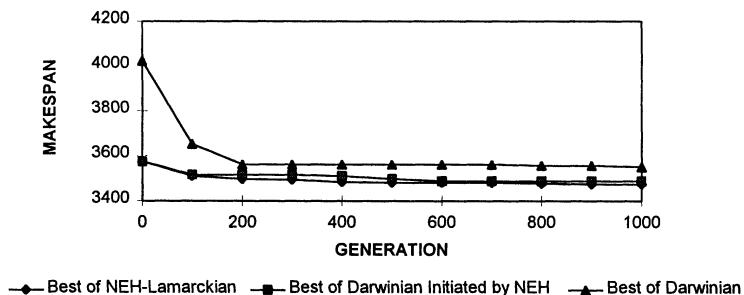


FIGURE 4.7 BEST SOLUTIONS FOUND BY DARWINIAN, DARWINIAN INITIATED BY CDS AND CDS-LAMARCKIAN GA

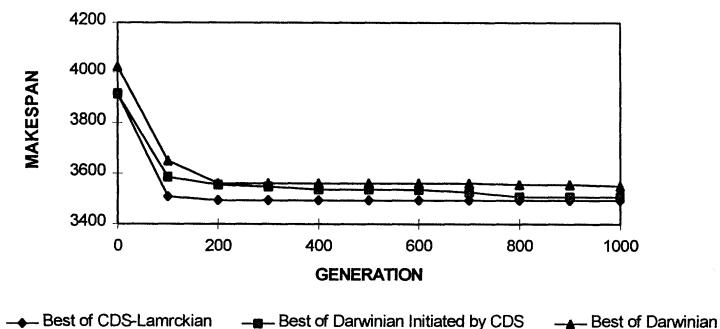
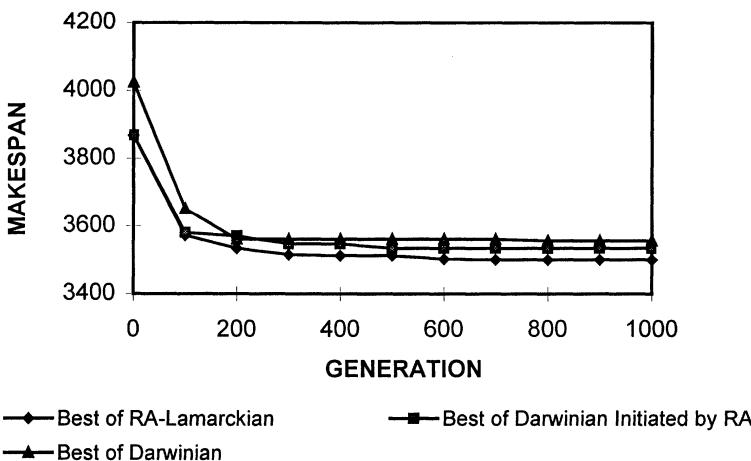


FIGURE 4.8 BEST SOLUTIONS BY DARWINIAN, DARWINIAN INITIATED BY RA AND RA-LAMARCKIAN GA



4.10 A MULTIOBJECTIVE GA FOR FLOWSHOP SCHEDULING

Murata, Ishibuchi and Tanaka (1996) provide perhaps the only GA-based approach available in the literature to produce a tentative set of Pareto optimal solutions to the problem combining the minimization of makespan, the minimization of total tardiness in a flowshop, and the minimization of total flow time. Their method uses a *weighted sum* of these multiple objectives $\{f_i(x)\}$ given by

$$f(x) = w_1 f_1(x) + w_2 f_2(x) + \dots + w_n f_n(x)$$

The weighted sum scheme by Murata et al. combines the individual objectives to form a scalar fitness function, which is then minimized by a simple GA. They use an elite preserve policy (preserving the best value solution for each individual objective). In addition, certain other tentative solutions are also preserved. Fitness of a solution is calculated from the weighted sum objective function.

The weights $\{w_i\}$ in this weighted sum are not the same for each solution, rather they are randomly varied to realize probing of various search directions. The investigators qualitatively compare their solutions to those obtained using VEGA (Schaffer, 1985; also see Section 8.1).

4.11 CHAPTER SUMMARY

This chapter has provided a synopsis of the common algorithms and heuristics used to solve two-machine and the general m -machine flowshop scheduling problems. These include Palmer's heuristic, Gupta's heuristic, the CDS heuristic, the RA heuristic and the NEH heuristic. This chapter also sketched several reported methods for applying genetic algorithms to sequence the flowshop. The different GA-based and hybridized approaches were recalled. Issues uncovered by researchers about starting solution choice, the choice of GA parameters and certain merits of hybridization were noted. A method recently proposed for using the GA to solve the multiobjective flowshop formulated as a scalarized single-objective (weighted sum) problem was also recalled.

Using life-size examples we established support for the following assertions about using GA to sequence flowshops:

- Given time, the Darwinian GA generally produces solutions superior to those produced by solution-generating heuristics such as Palmer, CDS, NEH or RA applied alone.
- However, for every *strategic* combination of solution methods tested (as indicated by Table 4.4), the pure Darwinian GA operating alone and started with a random initial population yields a statistically larger makespan.
- Whenever the initial population is given a "kick start" by filling it up with all-identical Palmer, CDS, NEH or the RA heuristic solutions, results improve.
- When Lamarckism is introduced by improving the best solution by "compressing" it, by invoking the Ho-Chang improvement heuristic, results improve further.
- An essential constituent in this process is the use of mutation to enable the process to avoid getting prematurely stuck.
- In almost every learning-incorporated case we should expect to produce solutions better than what is produced by random search or by the stand-alone application of the solution generating heuristic methods—Palmer, NEH, CDS and RA.

However, even if we recommend the use of learning-incorporated GAs, we do not for a minute downplay the value of *independently* developing solution-generating or solution-improving heuristics. Indeed both are highly valuable in speeding up genetic algorithms. Good solution-generating heuristic methods "heat up the engine block" quite effectively, so they assure a running start for the GA. The

solution-improvement methods such as the Ho-Chang heuristic, on the other hand, provide a shot of rich fuel to the GA when it appears to be "puttering."

Therefore, we strongly endorse continued research for both good solution-generating *as well as* solution-improving heuristic methods. We also recommend the development of innovative ways to incorporate Lamarckism, sharing, etc. to augment the effectiveness of the conventional (Darwinian) GA in shop scheduling.

We conclude this chapter by noting that GA seems to have been well exploited in the past five years to tackle the single-objective flowshop. The results produced by GA here are comparable to the better-performing heuristics devised to date.

The Boreland Turbo C++® code by Jain (Jain, 1999) included in the Appendix computes makespan using many of the above heuristic methods, the pure Darwinian GA as well as a Lamarckian GA incorporating the Ho-Chang heuristic. If you haven't worked with genetic algorithms before, you are invited to use this code and start your own experiments with flowshop scheduling by GA.

JOB SHOP SCHEDULING

Within the great variety of production scheduling problems that exist, the job shop scheduling problem (JSP) is one that has generated the largest number of studies. It has also earned a reputation for being notoriously difficult to solve. Nevertheless, the JSP illustrates at least some of the demands imposed by a wide array of real world scheduling problems. This chapter summarizes the well-known approaches—algorithmic and heuristic—to solve the *single objective* job shop problem. Attempts to tackle the multiobjective job shop are still relatively few.

5.1 THE CLASSICAL JOB SHOP PROBLEM (JSP)

The classical job shop differs from the basic flowshop in one important respect: flow of work in a job shop is not unidirectional. Because workflow is not unidirectional, inflows and outflows of work as shown in Figure 5.1 may characterize each machine in the job shop. Unlike the flowshop, in the job shop there is no initial machine that performs only the first operation of a job, nor is there is a terminal machine that performs only the last operation of the job.

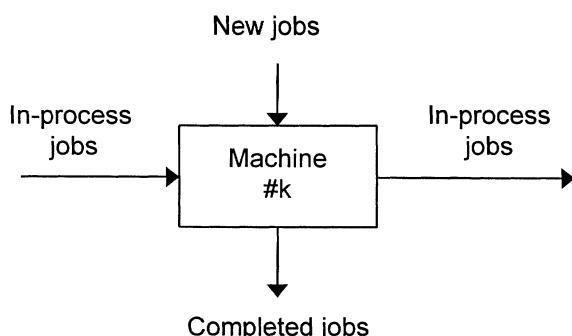


FIGURE 5.1 WORK FLOW IN A JOB SHOP

The classical job shop scheduling problem can be stated as follows: There are m different machines and n different jobs to be scheduled. Each job is composed of a set of operations and for a job, the *order* of these operations on the different machines is pre-specified, usually by processing or technological requirements. Each operation on a job is characterized by its own required machine and a corresponding fixed processing time. The restrictions in the *classical JSP* may be listed as follows

- No two operations of one job occur simultaneously.
- No pre-emption (i.e. process interruption) of an operation is allowed.
- No job is processed twice on the same machine.
- Each job is processed to its completion, though there may be waits and delays between the operations performed.
- Jobs may be started at any time; hence no release time exists.
- Jobs must wait for the next machine to be available.
- No machine may perform more than one operation at a time.
- Set-up times for the operations are sequence-independent and included in processing times.
- There is only one of each type of machine.
- Machines may be idle within the schedulable period.
- Machines are available at any time.
- The technological (usually related to processing) constraints are known in advance and are immutable.

The most widely used job shop management objective is to find a feasible schedule such that the completion time of the total production program (i.e. the makespan) is minimized. Feasible schedules may be obtained by permuting the processing order of operations on the machines (operation sequence) but without violating the technological constraints. Accordingly, this is a combinatorial minimization problem with constrained permutations of operations. More specifically, the operations to be processed on one machine form an *operation sequence* for that machine. Thus, the schedule for a JSP problem specifies the operation sequences of each machine involved.

Since each operation sequence can be permuted independently of the operation sequences of other machines, the total number of possible solutions for a JSP is $(n!)^m$, where n denotes the number of jobs and m denotes the number of machines involved. This total number constitutes the size of the solution space. The following illustration

shows how quickly the JSP can become too large to solve by conventional search methods and certainly to solve by enumeration.

Assume that the world is 20 billion years old. This means that the world is

$$20 \times 10^9 \times 365.25 \times 60 \times 60 = 6.31152 \times 10^{17} \text{ seconds old}$$

or 6.3×10^{23} microseconds old. Now, if we had a computer capable of producing and evaluating a job shop schedule every microsecond since the beginning of time, we could have produced only about $24!$ or 6.20×10^{23} schedules. In terms of a one-machine scenario, this means that since the beginning of time we could have optimally scheduled only 24 jobs in the job queue waiting for processing on a single machine. Garey and Johnson (1979) showed that the job shop scheduling problem is an *NP-hard* problem and even among all *NP-hard* optimization problems it is one of the least tractable.

In principle, there are infinite numbers of schedules possible for a job shop scheduling problem because superfluous idle time can be inserted between two operations. Alternatively, one may shift the operations to the left (on the time axis) to make the schedule as compact as possible.

A shift in the schedule is called a *local left-shift* if some operations can be started earlier in time without altering the operation sequence. A shift is a *global left-shift* if an operation can be started earlier in time without delaying any other operation. Based on these two concepts, three distinct schedules may be identified as follows:

1. Semi-active Schedule: A schedule is semi-active if no local left-shift exists.
2. Active Schedule: A schedule is active if no global left-shift exists.
3. Nondelay Schedule: A schedule is nondelay if no machine is kept idle at a time when it could begin processing some operations.

The set of constraints involved in real world applications is even more complex. In practice, only a few assumptions of the classical job shop scheduling problem may hold. However, in spite of the restrictive assumptions stated above, the JSP is already a notoriously hard scheduling problem to solve. Not surprisingly, therefore, the job shop scheduling problem remains popular in academic research,

particularly as the testbed for different solution techniques and innovations to solve complex combinatorial problems (see, for instance, Zweben and Fox, 1994).

Research on job shop scheduling has mostly addressed the *single objective* JSP (e.g. minimization of makespan) and there has been relatively little research on problems with multiple objectives. The multiobjective JSP is extremely difficult to solve. Note, however, that the multiobjective JSP (next to open shops) perhaps represents the true requirement of the real world of manufacturing and service delivery.

A comprehensive review of the algorithms and heuristic methods available to solve the single objective JSP is given in Gen and Cheng (1997). In the following subsections some of these methods are highlighted. In the description that follows, m and n denote respectively the number of machines present in the shop and the number of jobs to be processed.

5.1.1 INTEGER PROGRAMMING MODEL FOR JSP

Theoretically speaking, the job shop scheduling problem can be formulated as an integer programming problem (Greenberg, 1968; Baker, 1974; Cheng, Gen and Tsujimura, 1996). Such an approach is a general purpose solution approach. We describe the formulation using the following notations:

- m : Number of machines available to process the jobs
- n : Number of jobs to be scheduled
- t_{ijk} : Processing time of operation j of job i on machine k
- x_{ik} : Completion time of job i on machine k
- y_{ipk} : Indicator variable
- k_i : Machine at which the last operation of job i is scheduled
- H : A large positive number

The formulation relies on indicator variables to specify operation sequence. Let x_{ik} denote the completion time of job i on machine k (i.e. the completion time of the particular operation of job i that requires machine k). The time spans $\{x_{ik}\}$ are decision variables and their values will essentially determine the schedule. Next we write certain

inequalities representing precedence constraints as follows. Suppose that operation j of job i requires machine k and operation $(j-1)$ of job i requires machine h . Then in order for a set of x_{ik} to be feasible, it is necessary to have

$$x_{ik} - t_{ijk} \geq x_{ih} \quad 1 \leq j \leq m, 1 \leq i \leq n$$

where for the first operation ($j = 1$) the constraint is simply

$$x_{ik} - t_{ijk} \geq 0 \quad 1 \leq i \leq n$$

In addition, it is necessary to employ a large number of constraints to assure that no two operations are processed simultaneously by the same machine. Suppose, for example, that job i precedes job p on machine k , which means that operation (i, j, k) is completed before operation (p, q, k) begins. Then it is necessary to have

$$x_{pk} - t_{pqk} \geq x_{ik}$$

On the other hand, if job p precedes i on machine k , then it is necessary to have

$$x_{ik} - t_{ijk} \geq x_{pk}$$

Such constraints are called *disjunctive constraints* because one or the other alone must hold. In order to accommodate these constraints in the formulation, an indicator variable y_{ipk} is defined as follows:

$$y_{ipk} = \begin{cases} 1 & \text{if job } i \text{ precedes job } p \text{ on machine } k \\ 0 & \text{otherwise} \end{cases}$$

Then the constraints become

$$x_{pk} - x_{ik} + H(1 - y_{ipk}) \geq t_{pqk}$$

$$x_{ik} - x_{pk} + Hy_{ipk} \geq t_{ijk}$$

where H represents a very large positive number. For the mean flowtime minimization problem the entire formulation becomes

Minimize	$\sum_{i=1}^n x_{k_i}$
Subject to	$x_{ik} - t_{ijk} \geq x_{ih} \quad \forall (i, j - 1, h) \in (i, j, k)$ $x_{pk} - x_{ik} + H(1 - y_{ipk}) \geq t_{pjk} \quad 1 \leq i, p \leq n, 1 \leq k \leq m$ $x_{ik} - x_{pk} + H.y_{ipk} \geq t_{ijk} \quad 1 \leq i, p \leq n, 1 \leq k \leq m$ $x_{ik} \geq 0, y_{ipk} \geq 0 \text{ or } 1$

where k_i denotes the machine at which the last operation of job i is scheduled. The total number of variables in the above formulation is $mn(n+1)/2$. Hence for a 10-job 5-machine JSP this formulation would require specification of 500 constraints and 275 decision variables. This indicates why the integer programming model of the JSP rapidly becomes unsuitable as the problem size grows.

5.1.2 LINEAR PROGRAMMING MODEL FOR JSP

The formulation uses following notations:

- N : Set of operations
- M : Set of machines
- A : Set of pairs of operations constrained by precedence relations for each job
- E_k : Set of pairs of operations to be performed on machine k and which therefore cannot overlap in time
- t_i : Processing time of operation i
- d_i : Start time of operation i

Let $N = \{0, 1, 2, \dots, n\}$ denote the set of operations where 0 and n are considered as the dummy operations "start" and "finish," let $M = \{1, 2, \dots, m\}$ denote the set of machines, let A denote the set of pairs of operations constrained by precedence relations for each job, and let E_k denote the set of pairs of operations to be performed on machine k and which therefore cannot overlap in time. For each operation i , its processing time t_i is fixed, and the *start time* of the operation (d_i) is a variable that has to be determined during the optimization. Hence, the JSP may be formulated as follows:

$$\begin{array}{ll}
 \text{Minimize} & d_n \\
 \text{Subject to} & d_j - d_i \geq t_i \quad (i, j) \in A \quad (5.1) \\
 & d_j - d_i \geq t_i \text{ or } d_i - d_j \geq t_j \quad (i, j) \in E_k, k \quad (5.2) \\
 & t_i \geq 0
 \end{array}$$

The objective is to minimize the makespan. Constraint set (5.1) ensures that the processing sequence of operations for each job corresponds to the prescribed order. Constraints (5.2) ensure that each machine can process only one job at a time.

5.1.3 THE DISJUNCTIVE GRAPH MODEL FOR JSP

The notations shown below follow Balas (1969) and Sen and Gupta (1984).

- N : Nodes representing all operations
- A : Arcs connecting consecutive operations of the same job
- E : Disjunctive arcs connecting operations to be processed by the same machine
- (i, j, k) : Operation j of job i on machine k

The methodology of disjunctive arcs to solve the JSP was first proposed by Balas (1969). Balas defined a *disjunctive graph* G by the notation (N, A, E) where N is a set that contains nodes representing all operations, A is a set containing arcs that connect consecutive operations to be performed on the same job, and set E contains disjunctive arcs connecting operations to be done by the same machine.

A disjunctive arc can be "settled" (i.e. a decision can be made about which of the two operations that it connects should be done first) by choosing either of its two possible orientations. The construction of a schedule will settle the orientations of all disjunctive arcs to determine the sequence of operations on the same machine. Once the sequence is determined for a machine, the disjunctive arcs connecting operations to be processed by the machine will be replaced by the usual (oriented) precedence arrows, or *conjunctive* arcs. A set of disjunctive arcs E may be decomposed into cliques E_k , $E = E_1 \cup E_2 \cup E_3 \dots \cup E_m$, one clique for each machine.

Figure 5.2 (a, b and c) illustrate the disjunctive graphs for the two-job two-machine problem where each job consists of two operations as shown in Table 5.1. Figure 5.2(a) shows the precedence relationships. The notation used for an operation is (i, j, k) which identifies it as operation i for job j to be done on machine k . The construction of a schedule will resolve the question of whether operation $(1, 1, 1)$ will precede $(2, 2, 1)$ on machine 1 and whether $(1, 2, 2)$ will precede $(2, 1, 2)$ on machine 2. In Balas's terminology, these unresolved aspects of sequences are represented by *disjunctive* arcs, drawn as unoriented arrows (dotted lines) as shown in Figure 5.2(b). The disjunctive arcs represent precedence relations that cannot be determined before a schedule is constructed.

However, once a sequence is determined for machine 1, the disjunctive arc connecting nodes $(1, 1, 1)$ and $(2, 2, 1)$ will be replaced by the usual (oriented) precedence arrow or *conjunctive* arc. Similarly, the choice of a sequence for machine 2 will resolve the disjunctive arc between $(1, 2, 2)$ and $(2, 1, 2)$. The only restriction governing the use of these disjunctive arcs is that a resolution (i.e. replacement of a disjunctive arc by a conjunctive arc) must not form a cycle (or loop) in the network. Of the four potential resolutions of the network in Figure 5.2(b), only the one shown in Figure 5.2(c) contains a loop and cannot be interpreted as representing a valid schedule (note that there is no initial operation in this network).

TABLE 5.1 PROCESSING TIMES FOR A JSP

		Machine Operation	
		1	2
Job 1		1	2
Job 2		2	1

Even though the methodology of disjunctive arcs was proposed by Balas (1969), Balas himself later conceded the limitation of this method to solve complex and realistic JSFs (Bhatnagar, 1996).

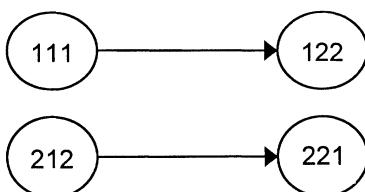


Figure 5.2(a)

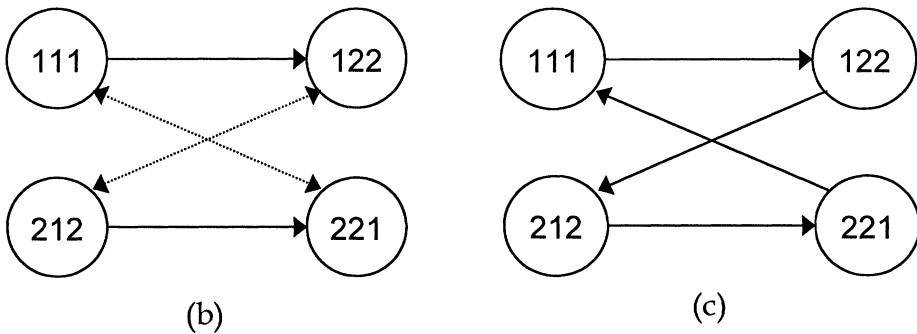


FIGURE 5.2 DISJUNCTIVE GRAPH REPRESENTATION OF A 2-MACHINE 2-JOB JSP

5.2 HEURISTIC METHODS FOR SCHEDULING THE JOB SHOP

As already mentioned, job shop scheduling is often an everyday practical problem in many factories. Since the problem is among the hardest of combinatorial problems to solve, it would be natural to look here for approximation or heuristic methods that could produce an acceptable schedule within a reasonable span of time. We use the following notation to describe these methods:

- m : Number of machines
- n : Number of jobs
- PS_t : A partial schedule containing t scheduled operations
- S_t : The set of schedulable operations at stage t , corresponding to a given PS_t
- σ_i : The earliest time at which operation $i \in S_t$ could be started
- ϕ_t : The earliest time at which operation $i \in S_t$ could be completed

All heuristic scheduling methods available in the contemporary literature may be classified into two broad groups. These are

- 1) One-pass heuristic, and
- 2) Multipass heuristic

A one-pass heuristic simply builds up a single complete solution by fixing one operation in the schedule at a time using *priority dispatching rules*. A one-pass heuristic may be used repeatedly to build the more capable multi-pass heuristic, albeit with some increase in computational effort.

A large number of dispatching rules are now available. These rules guide the selection of the operation to be scheduled from a specified subset of operations to be scheduled next. One reason for the popularity that dispatching rules enjoy is that such rules are usually easy to apply. Some are as follows.

5.2.1 PRIORITY DISPATCHING HEURISTICS FOR JSP

"Priority" rules are among the most frequently applied heuristics for solving scheduling problems because of their ease of implementation and their low computational complexity. The algorithms of Giffler and Thompson (1960) may be considered as the common basis of almost all priority rule-based heuristics.

Giffler and Thompson proposed two algorithms to generate a schedule named respectively as the *active schedule generation* procedure and the *nondelay schedule generation* procedure. As stated earlier, an active schedule has the property that no operation can be started earlier than when the (active) schedule specifies, without delaying another job. On the other hand, the nondelay schedule has the property that no machine remains idle if a job is available for processing.

The schedule generation procedure itself is a tree-structured approach. The nodes in the tree correspond to partial schedules, the arcs represent the possible choices, and the leaves of the tree are the set of enumerated schedules. For a given partial schedule, the algorithm first identifies all processing conflicts (i.e., operations competing for the same machine) and then an enumeration procedure is used to resolve these conflicts in all possible ways at each stage of partial schedule generation. In contrast to this enumerative approach, suitable heuristics that incorporate *priority dispatching* rules may be used to resolve such conflicts. That is, in the heuristic itself, one may specify a priority rule for selecting one operation among the conflicting operations.

The active and nondelay schedule generation procedures operate with a set of schedulable operations at each stage. "Schedulable" operations are the unscheduled operations with immediately scheduled predecessors, a set that can be simply determined from the precedence

structure of the operations. The number of stages for a one-pass procedure is equal to the total number of operations ($m \times n$). At each stage, one operation is selected to be added into the *partial schedule*. Conflicts among the operations are resolved by using priority dispatching rules.

For a given active partial schedule, the potential start time σ_i is determined by the completion time of the direct predecessor of operation i and the latest completion time on the machine required by operation i . The larger of these two quantities is σ_i . The potential finishing time ϕ_i is simply $\sigma_i + t_i$, where t_i is the processing time of operation i . Two priority dispatching procedures are outlined below.

A Priority Dispatching Heuristic for Active Schedule Generation

- Step 1. Let $t = 0$ and begin with PS_t as the null partial schedule (for notations see page 114). Initially S_t would include all operations with no predecessors.
- Step 2. Determine $\phi_t^* = \min_{i \in S_t} \{\phi_i\}$ and the machine m^* on which ϕ_t^* could be realized.
- Step 3. For each operation $i \in S_t$ that requires machine m^* and for which $\sigma_i < \phi_t^*$, calculate a *priority index* according to a specific priority rule. Find the operations with the smallest index and add this operation to PS_t as early as possible, thus creating a new partial schedule PS_{t+1} .
- Step 4. For PS_{t+1} update the data set as follows:
 - a) Remove operation i from S_t .
 - b) Form S_{t+1} by adding the direct successor of operation i to S_t .
 - c) Increment t by one.
- Step 5. Return to Step 2 until a complete schedule is generated.

A Priority Dispatching Heuristic for Nondelay Schedule Generation

- Step 1. Let $t = 0$ and begin with PS_t as the null partial schedule. Initially S_t includes all operations with no predecessors.
- Step 2. Determine $\sigma_t^* = \min_{i \in S_t} \{\sigma_i\}$ and the machine m^* on which σ_t^* could be realized.

- Step 3. For each operation $i \in S_t$ that requires machine m^* and for which $\sigma_i < \sigma_i^*$. Calculate a priority index according to a specific priority rule. Find the operations with the smallest index and add this operation to PS_t as early as possible, thus creating a new partial schedule PS_{t+1} .
- Step 4. For PS_{t+1} update the data set as follows:
- Remove the operations i from S_t .
 - Form S_{t+1} by adding the direct successor of operation i to S_t .
 - Increment t by one.
- Step 5. Return to Step 2 until a complete schedule is generated.

The remaining problem is to identify an effective priority rule.

The details of the above procedures may be found in Panwalker and Iskander (1977), Blackstone, Phillips and Hogg (1982), or Haupt (1989).

5.2.2 RANDOMIZED DISPATCHING HEURISTIC FOR JSP

While one-pass heuristics limit themselves to constructing a single solution, multipass heuristics (also called search heuristics) try to obtain better solutions by generating many of them, usually at the expense of a much higher computational effort. Techniques such as the branch-and-bound method and dynamic programming can guarantee an optimal solution. However, these are not practical for large-sized problems.

Randomized heuristics were an early attempt to provide reasonably accurate solutions for the JSP (Baker, 1974). The idea of a randomized dispatch is to start with a *family* of dispatching rules. At each selection opportunity (of an operation) the dispatching rule is chosen randomly and this process is repeated throughout an entire schedule generation. The process may be repeated several times and the consequent best result chosen as the answer. A randomized dispatching procedure is summarized below.

A Randomized Dispatching Heuristic for Active Schedule Generation

Baker (1974) describes this heuristic as follows.

- Step 0. Let the best schedule (BS) be a null schedule.
- Step 1. Let $t = 0$ and begin with PS_t be the null partial schedule. Initially S_t includes all operations with no predecessors.
- Step 2. Determine $\phi_t^* = \min_{i \in S_t} \{\phi_i\}$ and the machine m^* on which ϕ_t^* could be realized.
- Step 3. Select a dispatching rule randomly from the family of rules. For each operation $i \in S_t$ that requires machine m^* and for which $\sigma_i < \phi_t^*$, calculate a priority index according to a specified rule. Find the operation with the smallest index and add this operation to PS_t as early as possible, thus creating a new partial schedule PS_{t+1} .
- Step 4. For PS_{t+1} update the data set as follows:
- Remove the operations i from S_t .
 - Form S_{t+1} by adding the direct successor of operation i to S_t .
 - Increment t by 1.
- Step 5. Return to Step 2 until a complete schedule is generated.
- Step 6. If the generated schedule in the above step is better than the best one found so far, save it as BS . Return to Step 1 until iteration equals the predetermined number.

5.2.3 THE SHIFTING BOTTLENECK HEURISTIC FOR JSP

This heuristic, created by Adams, Balas and Zawak (1988), is presently regarded as the most powerful procedure among all heuristics for scheduling the JSP. It sequences the machines one by one, successively, taking each time the machine identified as a *bottleneck* among the machines not yet sequenced. Every time a new machine is sequenced, all previously established sequences are locally reoptimized. Both the bottleneck identification and the local reoptimization procedures are based on repeatedly solving a certain one-machine scheduling problem that is a relaxation of the original problem. The method of solving the one-machine problem is not new. However, Adams and his colleagues were able to speed up the time required for generating these problems considerably. Indeed, a key contribution of their approach is the way relaxation is used to decide upon the order in which the machines should be sequenced. The details of this scheduling procedure may be found in Adams et al. (1988), Applegate and Cook (1991) or Pinedo (1995).

5.2.4 THE ZWEBEN-FOX METHODS FOR SCHEDULING THE JOB SHOP

Scheduling may be viewed as an optimization process where limited resources are allocated over time among both parallel and sequential activities. Zweben and Fox (1994) describe a series of methods based on a constraint-directed approach to schedule the JSP. They describe several possible heuristic strategies including scheduling, rescheduling and iterative repair, a method that exploits constraint knowledge to converge the schedule being developed to near-optimal solutions.

5.3 GENETIC ALGORITHMS FOR JOB SHOP SCHEDULING

As noted earlier, the JSP is perhaps the most intractable one among the *NP-hard* class of optimization problems. The problem being one of combinatorial optimization, it is perhaps natural that one would at some point attempt to use genetic algorithms to solve it. Although the GA does not guarantee to resolve the optimization problem completely (i.e., GA does not assure us that it will find the globally optimal solution), it often serves as a powerful search procedure as it can sample a large search space randomly and efficiently. In fact, the assumption underlying the use of GA's for scheduling is that optimal solutions will be found in the neighborhood of good solutions. In other words, it is expected that exploitation of favorable features in sub-optimal solutions would eventually lead to the discovery of near-optimal solutions.

5.3.1 CHROMOSOME REPRESENTATION OF THE JSP SOLUTION

Anderson, Glass and Potts (1997) provide a review of the different methods to attack the JSP. Two well-written and exhaustive articles recently written on the use of GAs in job shop scheduling are by Cheng, Mitsuo and Tsujimura (1996a and 1996b). These authors called these papers rightfully "tutorials" on (1) the GA representation of the JSP solution, and (2) hybrid GA strategies to produce solutions efficiently. In the present and in the subsequent sections we recall the

essence of the material presented in those articles—to set the stage for solving the *multiobjective* JSP later in this text.

A rather important issue in building a GA for the JSP is to devise an appropriate chromosome representation of solutions, together with *problem-specific* "genetic" operations, so that all chromosomes generated in either the initial phase or the evolutionary process will produce feasible schedules. This is a crucial phase that affects all the subsequent steps in the GA. In recent years the following distinct representations for job-shop scheduling problem have been proposed

1. Operation-based representation (Bean, 1994).
2. Job-based representation (Holsapple, Jacob, Pakath and Zaveri, 1993).
3. Preference-list-based representation (Davis, 1985; Falkenaur and Bouffouix, 1991); Croce, Tadei and Volta, 1995; Kobayashi et al., 1995).
4. Job-pair-relation-based representation (Paradis, 1992; Nakano and Yamada, 1992a).
5. Priority-rule-based representation (Dorndorf and Pesch, 1995).
6. Disjunctive-graph-based representation Tamaki and Nishikawa, 1992).
7. Completion-time-based representation (Nakamo and Yamada, 1992b).
8. Machine-based representation (Dorndorf and Pesch, 1995).
9. Random key-based representation (Bean, 1994; Bean and Norman, 1995).

The solution representations above show considerable variety in them. These may be classified into the following two basic solution-encoding approaches:

- Direct approach
- Indirect approach

In the *direct approach*, a schedule is encoded directly into the chromosome, and genetic algorithms are used to evolve those chromosomes to discover a better schedule. The job-based representation, job-pair-relation-based representation, completion-time-based representation and random key-based representation belong to this class. In the *indirect approach*, one does not work with the schedule directly. For instance, for the priority-rule-based

representation, a representation of dispatching rules for job assignment is encoded into a chromosome and genetic algorithms are used to evolve more such chromosomes to determine a better sequence of dispatching rules. Subsequently, a schedule is constructed with the *sequence* of dispatching rules evolved. Precedence-list-based representation, priority-rule-based representation, disjunctive-graph-based representation and the machine-based representation belong to this class.

A brief description of these different representations is given below.

Operation-Based GA Representation

Gen, Tsujimura and Kubota (1994) devised this representation. The representation encodes a schedule as a sequence of operations, and each gene stands for one operation. All operations for a job are represented by the same symbol (e.g. "3" for job #3 or J_3) and they are interpreted according to the *order* of their occurrence in the sequence *in which they appear* in the chromosome. For example, for a three-job three-machine problem the representation would be as shown below where 1 stands for job J_1 , 2 stands for job J_2 , and 3 stands for job J_3 .

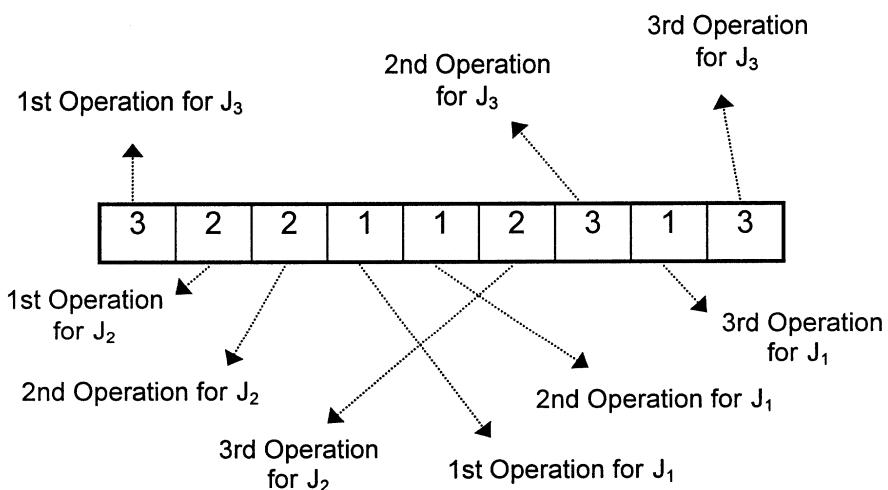


FIGURE 5.3 OPERATION-BASED REPRESENTATION OF A JSP SCHEDULE BY A GA CHROMOSOME

Job-Based GA Representation

This representation consists of a list of n jobs and the schedule is constructed according to the repeated sequence of jobs. For a given sequence of jobs, all operations of the first job in the list are scheduled first, and then the operations of the second job in the list are considered. The first operation of the job being scheduled is allocated the best available processing time on the machine the operation requires. Then the second operation is allocated a time slot, and so on, until the operations of the first job are all scheduled. The process is repeated with each of the jobs in the sequence, considered one by one. For example, for a three-job three-machine problem a chromosome is represented as

$$[2 \ 3 \ 1]$$

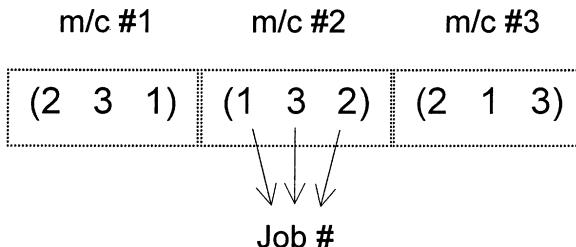
where 1 stands for job J_1 , 2 stands for job J_2 , and 3 stands for job J_3 . Any permutation of jobs corresponds to a feasible schedule. Holsapple, Jacob, Pakath, and Zaveri (1993) have used this representation to deal with static scheduling problems in a flexible manufacturing context.

Preference-List-Based GA Representation

This representation was first proposed by Davis (1985) for a particular kind of scheduling problem. Later, Falkenauer and Bouffouix (1991) used it to solve a job shop problem involving release times and due dates. Subsequently, Croce, Tadei, and Voltas (1995) applied it to tackle the classical JSP.

For the n -job m -machine JSP, a chromosome consisting of m sub-chromosomes is formed, one for each of the m machines. Each sub-chromosome is a string of symbols of length n . A symbol identifies an operation that needs to be done on the machine of interest. However, sub-chromosomes do not describe the operation sequence on the machine, rather, they are *preference lists* (preference orders). Each machine has its own preference list in which it will process the *jobs*. The actual schedule is deduced from the chromosome through a simulation, which analyzes the state of the waiting queues in front of the machine and, if necessary, use the preference lists to determine the schedule.

The machine of interest will choose the operation that appears first in the preference list. For example, for a 3-job 3-machine problem a chromosome is represented as shown below.



The chromosome shown in the above example consists of *three* genes— $(2 \ 3 \ 1)$, $(1 \ 3 \ 2)$ and $(2 \ 1 \ 3)$. The first gene $(2 \ 3 \ 1)$ is the preference list for jobs to be done on machine #1, $(1 \ 3 \ 2)$ is the preference list for jobs to be done on machine #2 and $(2 \ 1 \ 3)$ is the preference list for machine #3.

Job-Pair-Relation-Based GA Representation

Nakano and Yamada (1992b) use a binary matrix to encode a schedule. The matrix is determined according to the precedence relation of a pair of jobs on corresponding machines. A binary variable is defined to indicate the precedence relation for a pair of jobs. This representation is perhaps the most complex one and is highly redundant. Besides such complexity, the chromosomes produced here either by the initialization procedure or by genetic operations are often illegal. This slows down the GA.

Priority-Rule-Based GA Representation

Dorndorf and Pesch (1995) proposed a priority-rule-based genetic algorithm in which a chromosome is encoded as a sequence of dispatching rules for job assignment. Subsequently, a schedule is constructed with the priority dispatching heuristic based on the sequence of dispatching rules suggested by the chromosome.

Disjunctive-Graph-Based GA Representation

As mentioned in Section 5.1.3, the JSP can be represented by a disjunctive graph (Balas, 1969). Tamaki and Nishikawa (1992) proposed a disjunctive-graph-based GA representation, which can

also be viewed as a variety of job-pair-relation-based representations. The chromosome consists of a binary string corresponding to an order list of disjunctive arcs in E (the set of disjunctive arcs connecting the different operations to be processed by the same machine). Thus, this chromosome is not used to represent a schedule but it guides only a decision preference. A critical path-based procedure is used to derive a schedule. During this process, when a conflict of two nodes (operations) occurs on the machine, the corresponding bit of the chromosome is used to settle the processing order of the two operations—that is, to settle the orientation of the disjunctive arc between the two nodes.

Completion Time-Based GA Representation

Nakano and Yamada (1992b) proposed a completion time based representation. In this a chromosome is an ordered list of completion times of operations. For example, for a 3-job 3-machine problem a chromosome is represented as

C ₁₁₁	C ₁₂₂	C ₁₃₃	C ₂₁₁	C ₂₂₃	C ₂₃₂	C ₃₁₂	C ₃₂₁
------------------	------------------	------------------	------------------	------------------	------------------	------------------	------------------

where c_{jir} denotes the completion time for operation i of job j on machine r . Such representation is not suitable for most genetic operators because it will often yield an illegal schedule. Yamada and Nakano designed a special crossover operator for it.

Machine-Based GA Representation

Dorndorf and Pesch (1995) proposed a machine-based-genetic algorithm in which a chromosome is encoded as a sequence of machines and a schedule is constructed with a shifting bottleneck heuristic (Section 5.2.3).

Random Key-Based GA Representation

Bean (1994) was the first to use the "Random Key" representation. When this method is used, the genetic operations can produce feasible offspring without requiring additional overhead for "repair," for a wide variety of sequencing and optimization problems. Bean and

Norman (1995) successfully generalized this approach to the job-shop scheduling problem.

The random key representation encodes a solution with *random number*. These values are used as *sort keys* to decode the solution. For an n -job m -machine scheduling problem, each gene (a random key) consists of two parts: an integer in the set $\{1, 2, 3\dots m\}$ and a fraction generated randomly between 0 and 1. The integer part of any random key is interpreted as the machine assignment for that job. Sorting the fractional parts provides the job sequence on each machine. For example, suppose that for a 3-job 3-machine JSP a chromosome is represented as

1.34	1.09	1.88	2.66	2.91	2.01	3.23	3.21	3.44
------	------	------	------	------	------	------	------	------

For this chromosome the *first three keys* have 1 as their integer part, i.e., they represent the job order on the *machine 1*. The order of jobs on machine 1 is determined by sorting the three keys in the ascending order of their *fractional parts* (e.g. .34, .09 and .88 for machine 1). Thus, sorting the fractional parts of keys for machine 1 in ascending order gives the job sequence $2 \rightarrow 1 \rightarrow 3$. Similarly for machine 2 the sequence is $2 \rightarrow 3 \rightarrow 1$, and for machine 3 the sequence is $2 \rightarrow 1 \rightarrow 3$.

The job sequence thus produced using this representation may violate the precedence constraints. Therefore, accompanying this chromosome coding scheme, a pseudo-code is given by Bean and Norman for correcting and restoring precedence constraints.

5.3.2 HYBRID GENETIC SEARCH

When compared with conventional heuristics, genetic algorithms are not well suited for *fine-tuning* solutions that are very close to optimal solutions. Therefore, it frequently becomes essential that we incorporate conventional heuristics (such as local search) into genetic algorithms. This causes them to become competitive. Various methods of such *hybridization* have been proposed for the JSP. These may be classified into two basic approaches due, respectively, to Davis (1991) and Renders and Bersini (1994):

- Adapt genetic operators
- Incorporate conventional heuristic methods into genetic algorithms

The adaptation approach attempts to invent new genetic operators inspired by conventional JSP heuristics. Examples are the new crossover operator design based on the Giffler-Thompson algorithm (1960) or the new mutation operator designed by Cheng et al. (1996a) based on neighborhood search. The second approach involves hybridizing conventional heuristics where possible. This too may be done in a variety of ways such as

1. Incorporate the heuristic of value into GA *initialization* to generate a well-adapted initial population. Thus, a hybrid genetic algorithm with elitism can be guaranteed to do no worse than the conventional heuristic does.
2. Incorporate heuristics into the evaluation function to decode chromosomes.
3. Incorporate a local search heuristic as an add-on to the basic "loop" of the genetic algorithm (Holsapple et al., 1993), working together with mutation and crossover operators, to perform quick and localized optimization in order to *improve* offspring before returning it to be evaluated.

With the hybrid approach, the GA adopts the role of performing *global exploration* in the solution space while heuristic methods perform *local exploitation* around the available chromosomes. Because of the inherent complementary properties of genetic algorithms and conventional heuristics, the hybrid approach is seen to frequently outperform either method operating alone.

5.3.3 A GENETIC ALGORITHM BY GEN, TSUJIMURA AND KUBOTA

Chromosome Representation

Gen, Tsujimura, and Kubota (1994) adopted the operation-based representation for solving the job shop scheduling problem. As mentioned earlier, this chromosome representation encodes a schedule as a sequence of operations in which each gene stands for one operation. Gen and his coworkers name all operations of a job with identical symbols and then interpret them according to their order of occurrence in the sequence in a given chromosome. For an n -

job m -machine problem here a chromosome contains $n \times m$ genes. Each job appears in the chromosome exactly m times. The repeating occurrence of a job (in a gene) does not indicate a concrete operation of a job but it refers to an operation which is context-dependent. For example, for a three-job three-machine problem a chromosome is represented as shown in Figure 5.3.

You may observe that any *permutation* of the genes shown as segments in the chromosome above will always yield a feasible schedule without "repair." This is the key advantage of an operation-based JSP representation. Parent **p1** in Figure 5.4 contains a JSP schedule for a 4-machine 4-job job shop. **p1** contains exactly four "1"s, indicating the *four machine operations* required for job 1. Similarly, it contains four "2"s, four "3"s and four "4"s in it, thus it is a legal schedule.

Figures 5.4 through 5.7 show the steps of crossing such chromosomes and then legalizing the offspring, accomplished as described below.

Crossover

This method uses a *partial schedule exchange* crossover operator which considers the *partial schedules* (e.g. 4 1 2 4 in **p1**) in Figure 5.4 to be the natural building blocks in offspring in much the same manner as Holland (1975) has described his building blocks. A partial schedule is identified with the *same* job in the *first* and *last* positions of the partial schedule. The steps for the crossover operation are described below (refer to Figures 5.4, 5.5, 5.6 and 5.7).

In the illustration given for a 4-machine 4-job problem, we begin Step 1 with two parent chromosomes **p1** and **p2** (shown in Figure 5.4).

- Step 1. Pick one partial schedule (a substring of random length) in the first parent—randomly. The first position in this partial schedule is the 6th position. At this position job 4 is located (Figure 5.4).
- Step 2. Find the next-nearest job 4 in the same parent **p1**, which is in position 9. Thus the *partial schedule 1* formed is (4 1 2 4).
- Step 3. Locate the second partial schedule in parent **p2** as follows. It must be the first substring to begin and end with job 4 (as is true for first partial schedule). Thus partial schedule 2 is formed with the genes between the first operation of job 4

and the second operation of job 4 in parent **p2**. This partial schedule is 4 1 3 1 1 3 4 .

Step 4. Exchange the partial schedules 4 1 2 4 and 4 1 3 1 1 3 4 to produce progeny **o1** and **o2**. Figure 5.5 shows the result of this crossover exchange.

Usually, the partial schedules being exchanged would contain a different number of genes, so the offspring generated after exchanging might be illegal. An illegal offspring may not include or may have operations in excess of those required for each job. The "missed" and "exceeded" genes in an offspring indicate an illegal schedule as shown in Figure 5.6. The next step legalizes (repairs) each offspring by deleting exceeded genes and adding missed genes caused by exchange of partial schedules between **p1** and **p2**.

In the example, by crossover, offspring **o1** gained extra genes 3, 1, 1 and 3 while it lost gene 2. Therefore, the extra genes (3, 1, 1 and 3) are to be deleted, and the missing gene(s) (here 2) are to be inserted in **o1** (in the position immediately after the newly acquired partial schedule)—to make **o1** legal. Repair is then repeated for **o2** to render offspring **o2** also legal.

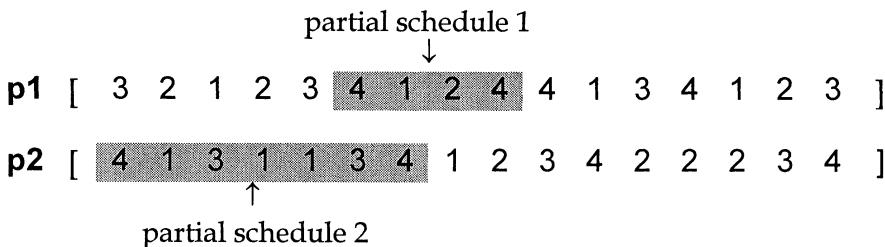


FIGURE 5.4 STEP 1: SELECT PARTIAL SCHEDULES

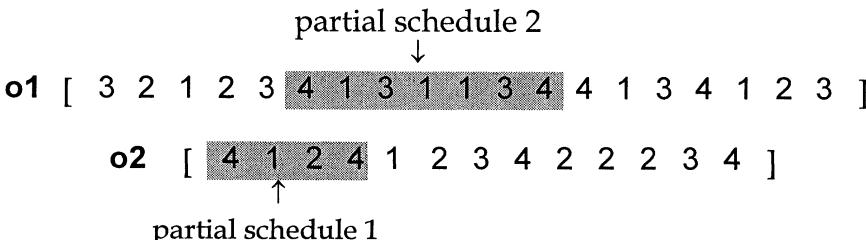


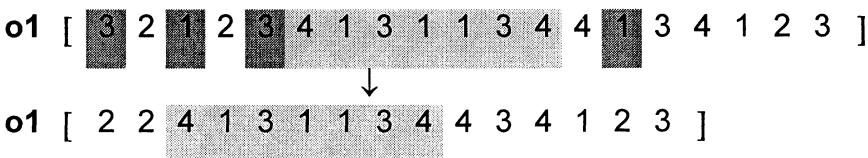
FIGURE 5.5 STEP 2: EXCHANGE PARTIAL SCHEDULES

Partial Schedule 1 4 1 2 4
 replaced in **p1** by 4 1 3 1 1 3 4
 \Rightarrow gene "2" missing in offspring **o1**, exceeded genes are "3," "1," "1" and "3."

Partial Schedule 2 4 1 3 1 1 4
 replaced in **p2** by 4 1 2 4
 \Rightarrow genes "3," "1," "1," "3," missing in offspring **o2**; exceeded gene is "2."

FIGURE 5.6 STEP 3: IDENTIFY MISSED/EXCEEDED GENES

Randomly Delete the exceeded genes "3," "1," "1" and "3" without disturbing the inserted partial schedule



Insert the missed gene "2" after the inserted partial schedule

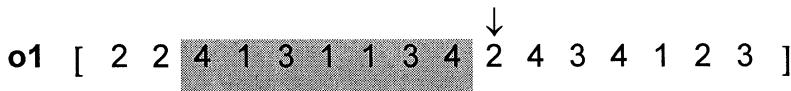


FIGURE 5.7 CROSSOVER STEP 4: LEGALIZE OFFSPRING o1

Mutation

The mutation operator used in this approach is the *job-pair exchange* mutation, that is, two non-identical jobs are picked randomly and then they exchange their positions as shown in Figure 5.8. The results are legal, hence no repair is necessary here.

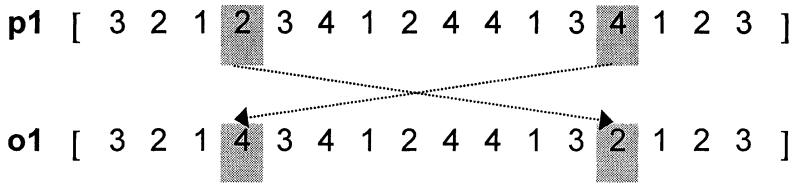


FIGURE 5.8 THE JOB-PAIR EXCHANGE MUTATION

5.3.4 A GENETIC ALGORITHM BY CHENG, GEN, AND TSUJIMURA

Cheng, Gen, and Tsujimura (1996a) modified the Gen-Tsujimura-Kubota (1994) method in order to enhance its efficiency. They made three major modifications:

1. A new decoding procedure was devised to guarantee generation of an active schedule.
2. A simplified crossover operation was proposed.
3. Mutation was designed with the neighborhood search technique and was used to perform an intensive search in order to find an improved offspring than that given by the previous job-pair exchange mutation.

5.3.5 A GENETIC ALGORITHM BY FALKENAUER AND BOUFFOUIX

Falkenauer and Bouffouix (1991) proposed an implementation of the genetic algorithm for the JSP with release times and due dates. They adopted the preference-list-based representation. Subsequently, Croce, Tadei, and Volta (1995) enhanced the Falkenauer-Bouffouix approach.

For the n -job m -machine classical job shop scheduling problem, a chromosome encoded as preference lists (Section 5.3.1) is formed from m subchromosomes; each sub-chromosome consists of n genes and is the preference list of operations for one machine. The actual schedule is deduced from the chromosome through a simulation based on the preference lists of operations to best satisfy the objective.

Falkenauer and Bouffouix also developed a variant of OX (order crossover) similar to "LOX" by Davis (1985). Note that the JSP chromosome is linear in contrast to the circular chromosome often used to solve the TSP (the travelling salesman problem). This is why TSP uses the OX crossover whereas JSP uses LOX. In JSP, LOX is applied to each sub-chromosome of the preference-list-based JSP chromosome independently. Falkenauer and Bouffouix used inversion as mutation operation whereas Croce, Tadei and Volta used the swap of two genes as mutation.

5.3.6 GENETIC ALGORITHM BY DORNDORF AND PESCH

Dorndorf and Pesch (1995) proposed two different implementations of GA for the job shop problem. The first uses the priority-rule-based representation while the other uses the machine-based representation. The common feature of these algorithms is that they both *hybridize* the GA with conventional heuristics to enhance its efficiency.

For the priority-rule-based representation, the investigators incorporated the well-known Giffler and Thompson (1960) algorithm into the genetic algorithm. Here GA is used to evolve a sequence of priority dispatching rules while the Giffler-Thompson algorithm is used to deduce a schedule from the encoded priority dispatching rules. For the machine-based representation, the investigators incorporated the well-known shifting bottleneck heuristic by Balas and Vazacopoulos (1994) into a GA. The GA here evolves a sequence of machines while the shifting bottleneck heuristic helps deduce a schedule from the encoded machine sequence.

Because these two encodings can be viewed as a permutation representation (such as the representation used to solve the travelling salesman problem), many conventional genetic operators may be applied here. The investigators used the scaling window technique (Hancock, 1994) to calculate fitness values of the chromosomes in order to increase selective pressure. They also used the elitist strategy (Goldberg, 1989) in selection so that the best solution in each generation always survived.

Uckun, Bagchi, Kawamura and Miyabe (1993) present an interesting study of variously representing the JSP solution to evolve it by GA.

They show that domain-specific chromosome representation, recombination (crossover) operators, and local enumerative search can significantly increase the GA's efficiency. They demonstrate that a simple GA combined with limited local search produces acceptable results in a short time, and performance improves even further when domain-specific knowledge is incorporated in the GA's design.

Among many others, two useful variants of GAs for the JSP are proposed by Fang, Ross and Corne (1993) and by Lin, Goodman and Punch (1997) respectively. Fang et al. employ a TSP-type representation and a performance enhancement method that uses dynamic sampling of the convergence rates in different parts of the genome. Lin et al. describe a hybrid GA approach consisting of a coarse-grain GA connected in a fine-grain GA topology. The fine-grain GA is imbedded in the coarse-grain GA. The authors indicate good convergence of this scheme on two benchmark JSPs. Storer, Wu and Vaccari (1992) indicate other GA methods for the JSP.

In the work on the multiobjective GA to be described in Chapter 10 we shall use the solution representation described in Section 5.3.3.

5.4 CHAPTER SUMMARY

In this chapter we have reviewed a number of approaches used by investigators to solve the *single objective* job shop problem. These include integer programming, linear programming, the disjunctive graph method, conventional heuristics and genetic algorithms. The heuristics reviewed include priority dispatching rules, the randomized dispatching heuristic and the shifting bottleneck heuristic. Genetic algorithms cited include various ways of representing the JSP schedule in a chromosome and the hybrid versions of GAs incorporating good, conventional heuristics in them.

Overall, as with the flowshop, a *hybrid* GA incorporating a good quality conventional heuristic makes an effective approach to solve the JSP. The method improves further if problem-specific knowledge is incorporated in solution representation and in the crossover operators.

6

MULTIOBJECTIVE OPTIMIZATION

As we pointed out in Chapter 1, rarely is a production manager interested *only* in getting the orders out the fastest way possible. Typically he/she also wants to minimize tardiness of the jobs from their committed shipping dates, maximize the use of expensive presses, furnaces, reactors and rolling mills, and human resources, minimize the mean flow time of jobs, etc. Such situations crop up everywhere and these are *multiobjective*. Ironically, however, most quantitative decision making methods confine to single objective optimization. These methods find only the *single best alternative* with respect to some figure of merit. In this chapter the subject of optimizing multiple objectives is discussed. Methods for solving multiobjective problems and the concept of *Pareto optimality* are reviewed. The chapter concludes with the citation of the nondominated sorting genetic algorithm (NSGA) (Srinivas and Deb, 1995), a new GA that rapidly discovers Pareto solutions.

6.1 MULTIPLE CRITERIA DECISION MAKING

In recent years the nature of decision problems has changed considerably. Serious doubts have been raised as to the adequacy of many classical models of decision situations and their solution techniques. For instance, numerous factors affect the typical modern industrial enterprise, leading not only to many decision variables but also to the multiplicity of *objectives* in decision making. Additionally, management objectives are also often conflicting. In a manufacturing enterprise, for example, apart from its overall objectives of profit and market share maximization, each department may have specific objectives to attain (Table 6.1). Such problems are called multiple-objective or multiple-criteria decision making (MCDM) problems.

TABLE 6.1 MULTIPLE MANAGEMENT OBJECTIVES IN A MANUFACTURING ENTERPRISE

Department	Objective(s)
Budget	Cost minimization
Production	<ul style="list-style-type: none"> • Production output maximization • Production time minimization • Resource utilization
Quality Control	<ul style="list-style-type: none"> • Product quality maximization • Rework minimization
Personnel	Minimization of hiring and firing
Marketing	Uninterrupted supply of products to customers

Many approaches to solve MCDM problems seem to deliberately and hence explicitly express organizational aspirations in terms of only a single criterion even though it is recognized that this may indeed be a misinterpretation of reality. In recent years, therefore, special *MCDM methodologies* have come into prominence. New tools and techniques have been developed for MCDM and subsequently applied to a variety of problems, the solutions of which control the success of many enterprises.

The general multiobjective problem requiring the optimization of k objectives simultaneously may be stated as follows.

$$\begin{array}{ll}
 \text{Max (min)} & Z_j = f_j(\mathbf{x}), \quad j = 1, 2, \dots, k \\
 \text{Subject to} & g_i(\mathbf{x}) \leq b_i, \quad i = 1, 2, \dots, m \\
 & \mathbf{x} \geq 0
 \end{array} \tag{6.1}$$

where \mathbf{x} is a p -dimensional vector of decision variables in the decision space X and $g_i(x)$ are certain inequality constraints.

In multi-criteria optimization, because the objectives $\{Z_j\}$ may be conflicting, there does not exist a single unique solution, which is globally maximum or globally minimum with respect to all the objectives. Increase in any one of these objectives may decrease the others and vice versa.

6.2 A SUFFICIENT CONDITION: CONFLICTING CRITERIA

A necessary condition of MCDM is the presence of *more than one* criterion. The sufficient condition is that the criteria must be *conflicting* in nature. Thus the MCDM problem can be defined as the problem with at least two conflicting criteria and for which there are at least two alternative solutions.

Criteria are said to be in conflict if the full satisfaction of one will result in impairing the full satisfaction of other(s). Two criteria are said to be "strictly" conflicting if increase in satisfaction of one results in a decrease in satisfaction of the other. The sufficient condition of MCDM, however, does not necessarily stipulate "strictly" conflicting criteria.

The currently available methods of MCDM have been based on the premise that all the objectives are dispensable and that all can be traded off although some may be more important than the rest.

6.3 CLASSIFICATION OF MULTIOBJECTIVE PROBLEMS

There are several ways to classify the different approaches to multiobjective optimization. Adulbhan and Tabucanon (1989) classified the techniques into three main approaches based on the way the initial multiobjective problem is transformed into a mathematically manageable format. These approaches are, respectively, (a) conversion of secondary objectives into constraints, (b) development of a single combined objective function, and (c) treatment of all objectives as constraints. Hwang, Masud, Paidy and Yoon (1982), on the other hand, propose grouping of techniques according to the *stage* at which the analyst needs information from the decision-maker. The classification is divided into four approaches: (a) no articulation of decision maker's preference data, (b) *a priori* articulation of preference data, (c) progressive articulation of preference data, and (d) *a posteriori* articulation of preference data.

Quite naturally, the solution methods and the underlying philosophies are materially different for these classes. Among those who have extensively surveyed the methods for multiobjective optimization are Johnson (1968), Roy (1971), Cochrane and Zeleny (1973), Lietmann and Marzollo (1975) and Hwang and Masud (1979).

6.4 SOLUTION METHODS

6.4.1 Single Objective Approach

A simple way of handling multiobjective optimization problems with k objectives would be to optimize one objective (the most "important" one) and to treat the resulting $k-1$ "secondary" objectives into constraints (Tabucanon, 1989). Specifying a maximum (for minimization) or minimum (for maximization) level of attainment may do this for each of the secondary objectives. Thus the multiobjective problem (6.1) is converted into and subsequently solved as a single objective optimization problem, as follows.

$$\begin{array}{ll} \text{Maximize} & Z_1 = f_1(\mathbf{x}) \\ \text{Subject to} & g_i(\mathbf{x}) \leq b_i, i = 1, 2, \dots, m \text{ (original constraints)} \\ & f_j(\mathbf{x}) \geq z_j^1, j = 2, 3, \dots, k \text{ (additional constraints)} \\ & \mathbf{x} \geq \mathbf{0} \end{array} \quad (6.2)$$

where z_j^1 are the pre-specified minimum levels of attainment required of the remaining ($k - 1$) objectives. In this all objectives are assumed to be of maximizing nature.

Although the above method is practicable, there are certain cases where the approach gives no defined feasible region after the introduction of the additional ($k - 1$) constraints.

6.4.2 Global Criterion Method

This method (Tabucanon, 1989) develops a *global* objective function which is made up of the ratio of sum of the deviations of the values of the individual objective functions from their respective *ideal* values $\{f_i(x^*)\}$ to that of their respective values $\{f_i(\mathbf{x})\}$. Thus, from the original k objective functions, a single function is formulated and the task tantamounts to solving a single objective optimization problem. This method does not require explicit information on the relative importance of the objectives. Additionally, there is less subjectivity involved in the formulation process. The problem becomes

$$\text{Minimize} \quad F = \sum_{l=1}^k \left[\frac{f_l(x^*) - f_l(\mathbf{x})}{f_l(x^*)} \right]^p \quad (6.3)$$

$$\begin{array}{lll} \text{Subject to} & g_i(\mathbf{x}) & \leq 0, \quad i = 1, 2, \dots, m \\ & \mathbf{x} & \geq 0 \end{array}$$

where $f_l(\mathbf{x}^*)$ is the value of objective function l at its individual optimum \mathbf{x}^* , $f_l(\mathbf{x})$ is the function itself, p is an integer-valued exponent that serves to reflect the importance of the objectives and g_i is the function of constraint i .

6.4.3 Utility Function Method

The utility function method (Tabucanon, 1989) converts the multiobjective optimization problem into a single objective problem as follows:

$$\begin{array}{lll} \text{Maximize} & Z = F[f_1(\mathbf{x}), f_2(\mathbf{x}), \dots, f_k(\mathbf{x})] & (6.4) \\ \text{Subject to} & g_i(\mathbf{x}) \leq 0, \quad i = 1, 2, \dots, m \\ & \mathbf{x} \geq 0 \end{array}$$

where F is the utility function of the multiple objectives representing the decision-maker's preferences. If F is properly determined, the solutions obtained will be expected to ensure the decision-maker's maximum satisfaction.

However, at times the determination of F can be extremely difficult. Further, F may appear in many forms, the most common forms assuming the decision-maker's utility function to be additively separable with respect to the objectives. The *additive utility function method* therefore converts the objective functions into one with the following form:

$$\text{Maximize} \quad Z = \sum_{j=1}^k w_j f_j(X) \quad (6.5)$$

Here w_j indicates the relative importance the decision-maker attaches to objective j and it must be specified for *each* of the k objectives *a priori*.

6.4.4 Minimum Deviation Method

This method (Tabucanon, 1989) is applicable when the analyst has information about the optimal values of the objectives but their relative importance is not known. The method aims at finding the best compromise solution which minimizes the sum of individual objectives' *fractional* deviations obtained from individual optimum values. The fractional deviation of an objective refers to the ratio of the deviation of a value of that objective from its individual optimal solution and its *maximum* deviation. The maximum deviation of an objective is obtained from the difference between its individual optimal solution and its least desirable solution, which would correspond to the individual optimal solution of one of the other objectives.

6.4.5 Goal Programming Approach

Goal Programming (GP) is a widely used multi-criteria method that requires both ordinal and cardinal information for multiple objective decision making (Tabucanon, 1989). In GP, *deviation variables* (from goals) with assigned priorities and weights are minimized instead of optimizing a single objective criterion directly as done, for instance, in linear programming. The general statement of programming is as follows:

$$\begin{aligned} \text{Minimize} \quad & Z = \sum_{i=1}^m (p_{oi}d_i^+ + p_{ui}d_i^-) \\ \text{Subject to} \quad & \sum_{j=1}^n a_{ij}x_j + d_i^- - d_i^+ = b_i, \quad i = 1, 2, \dots, m \\ & x_j, d_i^-, d_i^+ \geq 0, \quad i = 1, 2, \dots, m; \quad j = 1, 2, \dots, n \end{aligned} \quad (6.6)$$

where x_i are the basic variables in the goal equations, b_i are the targets or goals, a_{ij} are coefficients of the basic variables, d_i^- represents the underachievement of goal i , d_i^+ represent the overachievement of goal i , p_{ui} is the priority associated with d_i^- , and p_{oi} is the priority associated with d_i^+ . If overachievement is acceptable, d_i^+ can be eliminated from the objective function Z and if underachievement is acceptable, d_i^- can be eliminated. If goal i must be achieved exactly as

defined, then both d_i^- and d_i^+ must both appear in the objective function. If goals are classified in r ranks, priority factors p_r ($r = 1, \dots, m$) should be assigned to the deviation variables. The priority factors have the following relationships: $p_r \gg N p_{r+1}$, which implies that the multiplication of a quantity N , however large it may be, cannot make p_{r+1} greater than or equal to p_r . The algorithm uses a modified simplex (Tabucanon, 1989).

From the above it becomes evident that conventional MCDM methods essentially convert a multiple objective optimization problem into a single objective problem by one method or another. No method takes care of optimizing all the objectives simultaneously to obtain a *set* of solutions (such as the "Pareto optimal" or "efficient" solutions discussed later in this chapter) which offer a way to simultaneously "satisfice" all objectives (Simon, 1969). Recently, heuristic search methods that are based on the Pareto-optimality rationale to produce nondominated solutions to multiobjective problems have been proposed. One such heuristic is Nondominated Sorting Genetic Algorithm (NSGA), created by Srinivas and Deb (1995). NSGA uses niche formation and speciation as occurring in natural evolution. NSGA is actually a clever extension of Simple Genetic Algorithm (SGA), the original genetic algorithm created by Holland (1975) to optimize a single objective.

6.5 MULTIPLE CRITERIA OPTIMIZATION REDEFINED

A recently proposed method for treating the analytical phase of the MCDM process is called multiple criteria optimization or, in short, multiobjective optimization (Seo and Sakawa, 1988). According to this viewpoint, multiple criteria optimization contains two key concepts: (1) Pareto optimality and (2) the preferred decision (or preferred solution). In general, the *decisions* with Pareto optimality are not uniquely determined, unlike, for instance, what goal programming produces. In multiobjective optimization problems, there usually exist many solutions that are optimal in the *Pareto* sense, a concept put forth by economists and explained in the next section. Owing to such plurality of optimal decisions, the most desirable decision may be selected *after* one has generated the Pareto optimal or *nondominated* solutions. The final solution thus selected as the most desirable, or at least the best-compromised solution, is called the *preferred solution*.

6.6 THE CONCEPT OF PARETO OPTIMALITY AND "EFFICIENT" SOLUTIONS

In optimization, the "optimal" solution is one which attains maximum values (or minimum values) of all of several objectives simultaneously. Thus for a multiobjective problem the solution x^* is optimal if and only if $x^* \in s$ where s is the feasible region and $f_i(x^*) \geq f_i(x)$ for all i and for all $x \in s$. However, in case of multiobjective optimization with conflicting objectives, there is *no* unique optimal solution. A simple optimal solution may exist here only when the objectives are non-conflicting. Thus for conflicting objectives one may at best obtain what economists call "efficient" or nondominant solutions (Figure 6.1).

In such situations the notion of optimality is not obvious (Goldberg, 1989). Indeed if one refuses to interrelate the relative values of the different objectives *a priori* (what Goldberg calls comparing apples and oranges), then one must find a different definition of optimality—one that respects the soundness of each of the objectives. Economists achieved this rationality by introducing the idea of Pareto optimality.

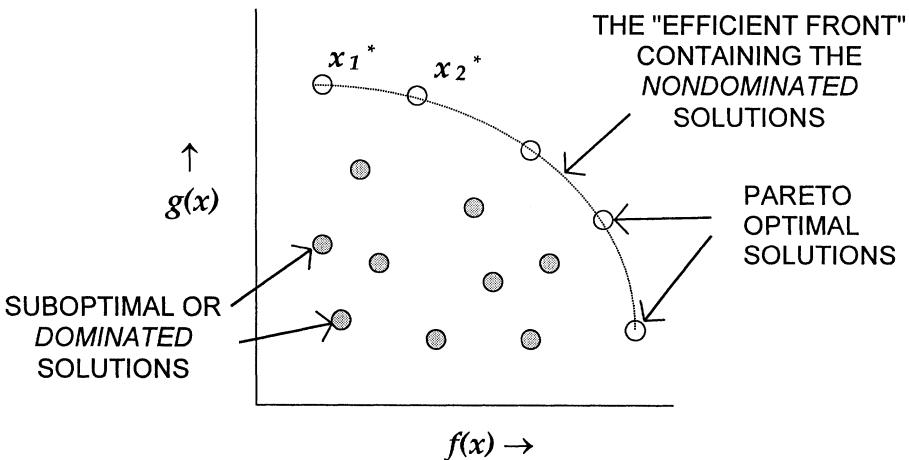


FIGURE 6.1 THE EFFICIENT FRONT IN A BI-OBJECTIVE MAXIMIZATION PROBLEM

An *efficient* solution (also called a *Pareto optimal* solution) is one in which no increase can be obtained in any of the objectives without

causing a simultaneous *decrease* in at least one of the remaining objectives (Keeney, 1983). For the bi-objective problem displayed in Figure 6.1, each solution x_i^* on the efficient front is such that no other feasible solution x exists such that $f(x) \geq f(x_i^*)$ and $g(x) \geq g(x_i^*)$. For a multiobjective problem solution, x_i^* is efficient if and only if there does not exist any $x \in s$ such that

$$f_j(x) \geq f_j(x_i^*) \quad \text{for all } j \text{ and}$$

$$f_j(x) > f_j(x_i^*) \quad \text{for at least one } j.$$

The *nondomination* property of solutions may be explained as follows. In a minimization problem, a solution vector $x^{(1)}$ is partially dominated by another vector $x^{(2)}$, (written as $x^{(1)} \prec x^{(2)}$), when no component value of $x^{(2)}$ is less than $x^{(1)}$ and at least one component of $x^{(2)}$ is strictly greater than $x^{(1)}$. In a minimization problem if $x^{(1)}$ is partially less than $x^{(2)}$, we say that the solution $x^{(1)}$ *dominates* $x^{(2)}$ or the solution $x^{(2)}$ is inferior to $x^{(1)}$.

Any member of such vectors that is not dominated by any other member is said to be *nondominated* or *noninferior*.

Thus, Pareto optimal solutions to a multiobjective optimization problem are the nondominated or "efficient" solutions. However, although the concept of Pareto optimality is now over 70 years old (Pareto, 1906), methods for finding nondominated solutions are relatively few. When the objective functions and the constraints are well behaved, the " ε -method" (Seo and Sakawa, 1988) (outlined below) can find the nondominated solutions. For other problems, search methods are still the only recourse.

6.6.1 The ε -Constraint Method to Produce Efficient Solutions

The multiple criteria optimization problem involving *well-behaved* (continuous and differentiable) objective functions and constraints (Seo and Sakawa, 1988) may be stated in the following form

$$\begin{aligned} \text{Max}_{x \in X} \quad & [f_1(x), f_2(x), \dots, f_m(x)] \\ \text{Subject to} \quad & g_j(x) \leq 0, \quad j = 1, 2, \dots, J \end{aligned} \tag{6.7}$$

The corresponding " ε -constraint" problem $P_k(\varepsilon)$ to discover Pareto optimal solutions here is formulated as follows.

$$\begin{array}{ll} \text{Max} & f_k(x) \\ x \in X & \end{array} \quad (6.8)$$

$$\begin{array}{l} \text{Subject to } f_i(x) \geq \varepsilon_i^1, \quad i = 1, 2, \dots, m, \quad i \neq k \\ \quad g_j(x) \leq 0, \quad j = 1, 2, \dots, J \end{array}$$

where parameter $\varepsilon_{i,r}$ is calculated as $f_{i,r} = f_{i,r-1} - \varepsilon_{i,r}$, $\varepsilon_{i,r} > 0$. i denotes the number of times of repetitive calculation ($r = 1, \dots, T$). $f_{i,0}$ is an initial value of the i -th objective function $f_i(x)$, which has been optimized independently for $x \in X$.

In this method, the maximization problem (6.7) having m objective functions is reduced to the scalar-valued maximization problem (6.8) having only the k^{th} objective function $f_k(x)$, in which all the remaining ($m-1$) objective functions are treated as constraints. The problem $P_k(\varepsilon)$ is repeatedly solved for $x_r \in X$ corresponding to the value of the constraint constant $\varepsilon_{i,r}$ which is parametrically varied by shifting the parameter $\varepsilon_{i,r} > 0$ in each iteration. The set of solutions obtained in each iteration derives the noninferior (Pareto optimal) solution set.

There is no established protocol for employing the ε -constraint method. The method is ad hoc because it is seldom precisely clear how one should configure the problem. Which objectives are to be converted to constraints at which ε values is largely determined by the user's sense of experimentation.

We note further that rather few objectives and constraints in problems in management science are *well behaved*, a precondition for the ε -constraint method to apply. Shop scheduling decisions involve sequences (of jobs, machines, etc.), hence the constraints and objectives of relevance here are most certainly not well behaved.

The fitness landscapes for *multi-criteria* scheduling problems are also not well behaved. However, novel GAs devised for such problems can now find their solutions with relative ease. Chapter 7 describes the theory of such methods. *Niche formation* and *speciation*—two phenomena found in nature that lead to co-existing life forms when resources are limited—form the foundation of these methods.

Chapter 7 recalls the biological moorings of genetic algorithms. We point out that evolution is not only biological, but also *cultural*. Indeed, much of the changes that we see around us is the product of learning. Additionally, we show how niche formation and speciation have led to much of the diversity in the natural world. Subsequently we show how these phenomena can enhance the power of the genetic algorithm.

NICHE FORMATION AND SPECIATION: FOUNDATIONS OF MULTIOBJECTIVE GAs

The genesis of GAs, as noted in Chapter 2, was an insightful deduction by several researchers that some aspects of natural evolution could be cast into useful algorithms to solve difficult global search problems in engineering, computer science and economics. Among them was John Holland who began his studies in the 1960s. About the same time, Bremermann (Bremermann, Roghson and Salaff, 1966) independently wrote that mutation, mating and selection of genotypes could be emulated to devise evolutionary computerized schemes that can seek out optima and converge very well. In 1975 Holland published a paradigm for an important process that he termed "adaptation." He outlined a "genetic plan"—an algorithm to seek out global optima in function optimization (Holland, 1975).

Aside from such attempts to discern adaptation and other life processes which occur at a "macro" level, much progress has been also made in explaining various "micro" processes with the Crick-Watson discovery (Watson and Crick, 1953) of the structure of the DNA molecule. DNA, we now know, plays a key role in transferring heredity. A good introduction to the subject is De Robertis and De Robertis Jr. (1987). The actual processes occurring in nature, however, are quite enigmatic and the debate about their details is still far from over (Dawkins, 1996). Even as this millennium closes, many scientists believe that we perhaps have only a "reasonable" grasp of how nature continually evolves organisms that better "fit" into their natural environment (Russell, 1998). Still, the study of certain aspects of natural evolution that we summarize in this chapter has already inspired the creation of some very effective genetic algorithms.

This chapter recapitulates the modern view of life processes and evolution. It then describes *niche formation* and *speciation*, two closely related phenomena frequently observed in nature that can endow the simple GA with multiobjective problem solving capability. Niches are distinct behavior patterns such as sharing of habitat, food, etc. that organisms develop as they struggle to compete with one another for limited resources. Niches also form as organisms attempt to survive in unfavorable environmental conditions. Speciation is the process by which *new* species evolve in nature, producing stable and coexisting diversity of organisms that do not interbreed. In the next chapter we describe how emulation of these two conditions has been the cornerstone in the design of special GAs that solve *multiobjective* optimization problems. These special GAs seek out Pareto optimal solutions.

7.1 BIOLOGICAL MOORINGS OF NATURAL EVOLUTION

Ecology is the branch of biology that deals with the relationships of organisms with their surroundings and with one another. Ecology studies the nature of environments and the special demands particular environments make upon organisms that inhabit them. Ecology also studies the characteristics of organisms, species and groups that promote their tolerance of environmental conditions. *Evolution* is the name given to the series of gradual changes as populations adapt to their changing surroundings (Young, 1971). Over the past two hundred years various theories have been proposed about how evolution has occurred in nature. However, today it is held that no theory of evolution can be acceptable if it fails to account for a key characteristic of living organisms—their ability to gradually *adapt* to their environment. Formally, adaptation is defined as the possession by organisms of characteristics that suit them to their environment or their way of life (Magill, 1991).

The modern theory of evolution is based on the observations and deductions made by Darwin and his contemporaries. Both Lamarckian and Darwinian theories (Section 2.1) moor on adaptation—the process whereby a structure is progressively modified to give better performance in its environment. Lamarckian adaptation is based upon changes in an individual that it acquires during its lifetime, directly by increased use or disuse of organs in

response to "a need that continues to make itself felt." An example is the ironsmith's acquiring strong right biceps. Lamarckism holds the view that these changes or adaptations are inherited by the offspring to cause evolution.

Consequently, the Lamarckian explanation of evolution has been called "inheritance of acquired characters." By contrast, Darwin's view of adaptation accepts life as an active equilibrium between the living organism and its surrounding environment. This equilibrium can be maintained only if the environment suits the organism, which is then said to be "adapted" to that environment. Darwin's theory of evolution is affixed on natural selection, a process in which the well-adapted individuals survive to procreate while the least fit die off. Thus, the combined processes of natural selection and inheritance constantly adjust the population to a changing environment. Darwin said that natural selection, assisted by local adaptation, mechanizes evolution.

Modern biology is gradually finding stronger and stronger support that natural selection and genetic variation together could actually have caused evolution. On the other hand, biologists have found little evidence till now to support the "acquired characteristics" or Lamarckian theory of evolution. As we explain later, Lamarckism is incompatible with the "Central Dogma" of molecular biology, a belief originally put forth by Francis Crick in 1958 and held by many of today's biologists (Russell, 1998). The Central Dogma states that the flow of *genetic information* can only occur from DNA to RNA to protein and not in reverse. The explanation is given as follows.

The physical features or observable behavior of an organism is called its "phenotype." Studies have now confirmed that phenotypic characteristics of an organism are determined by (a) genetic information passed on to the progeny organism by its parents, and/or (b) characteristics resulting from phenotypic reactions in the organism to environmental stimuli. Natural selection genetically propagates many of the selected parents' abilities—including their survival capability in some particular environment—to the progeny. However, the effectiveness of natural selection in improving the adaptation of a population to its environment depends on the extent to which the differences between individuals, which are responsible for their success or failure in the struggle for existence, are *inherited* by their progeny.

Beginning with Gregor Mendel's results of meticulously conducted pea plant breeding experiments that he presented in 1865, cytologists and geneticists have been studying the phenomenon of *inheritance* (collection of traits that are passed on from parents to progeny) constantly. Stahl (1964) gives an early and lucid introduction to the subject. Much assistance was provided in the understanding of inheritance by the invention of the microscope. This device, besides demonstrating the cellular basis of life, also revealed the cell nucleus, the chromosomes within it, and the processes of *mitotic* and *meiotic* cell divisions (defined below) and fertilization—the key actors of inheritance. We have now established that inheritance resides in the organism's *DNA* (deoxyribonucleic acid), a long nucleic acid molecule consisting of various arrangements of four kinds of smaller, distinct molecules (Russell, 1998).

The evidence came initially from bacterial transformation and viral infection experiments. In the case of viral infection it was shown by radioactive labeling experiments that the ability to infect and give rise to more viral particles rests with the DNA and not with proteins. Thread-like structures of DNA, called *chromosomes*, contain the genetic grist of the parents and act as the carriers of genetic information. Inheritance (parental traits passed on to progeny) initially gets coded in the parents' chromosomes where it is held in smaller molecules or segments called "genes," a unit of heredity composed of DNA. Thus, the explanation for inheritance that we accept today is that the material translated from generation to generation is not the form and substance of a pterodactyl or a mammoth, but primarily the capacity to *synthesize* particular proteins.

A very brief description and the support for this viewpoint are as follows.

A *gene* is the shortest length of the chromosome that cannot be broken by recombination (Section 2.1). Genes have two functions. They serve as templates for making accurate copies of themselves when cells multiply. Genes also contain and provide the *genetic codes* (particular sequences of *nucleotides*—certain unique nitrogen containing organic compounds) for the synthesis of the amino acid chains of a particular protein. A deeper explanation says that a gene determines the formation of an *enzyme* (a specific protein molecule which acts as a catalyst for some particular and complex chemical reaction) needed to synthesize some particular protein in our body. Thus, a *defined*

sequence of nucleotides is the means by which genetic information in DNA controls the manifestation or synthesis of specific proteins by the cell containing the chromosome.

The genetic endowment (called the *genotype*) of an individual is comparable to a store of information or instructions that, interacting with the environment in which the individual lives, determines the individual's *phenotype* (the outward appearance or observable characters of the organism) and its development. In other words, an organism's genotype determines its proteins, which in turn determine its phenotype.

In 1944 it was discovered by a series of extensive experiments by Oswald Avery and his colleagues that DNA is the genetic material that carries inheritance (see Smith, 1993, page 68). DNA is organized in the form of thousands of different kinds of genes, each of which has a coding capacity of one unit of genetic information. Furthermore, it was found that DNA is a major constituent of the chromosomes within the cell nucleus and it plays a central role in determining the hereditary characteristics by controlling protein synthesis in cells. The DNA molecule consists of two chains of nucleotides wound round each other and linked together by hydrogen bonds to form a spiral (double helix) ladder-like shape (Watson, 1968). When a cell divides, its full length of DNA also replicates in such a way that each of the two daughter DNA molecules is identical to the parent DNA.

Another nucleic acid called RNA (ribonucleic acid) is also present in living cells. RNA is concerned with protein synthesis and it comes in several varieties—*messenger*, *ribosomal*, *transfer*, and *soluble*. The special RNA known as *messenger* RNA (or *m*-RNA) is responsible for carrying the genetic code transcribed from DNA to sites within the cell where proteins are synthesized.

In summary, the *structure, function, development* and *reproduction* of an organism, as we understand these today, depend on the properties of the *proteins* present in each cell and tissue of the organism (Russell, 1998). A protein consists of one or more chains of amino acids. Each chain of amino acids is called a polypeptide. The sequence of amino acids in a polypeptide chain is coded for by a gene, i.e., a specific base-pair sequence in DNA. When a particular protein is needed in a cell, the genetic code for that protein's amino acid sequence must be read from the DNA and processed into the finished protein. The two steps

occurring here are (a) the transfer of information from the template DNA molecule to a RNA molecule (transcription), and (b) protein synthesis (translation)—conversion of the messenger (*m*-RNA) base sequence information into the amino acid sequence of the target polypeptide.

The genetic process of inheritance may be described in short as follows. A cell is the structural and functional unit of life. Each cell consists of a mass of protein material that is differentiated into a jelly-like substance and a nucleus, which contains DNA. Bacterial cells are primitive in the sense that in these a membrane does not bound the nuclear material and reproduction occurs by simple cell cleavage. In the more evolved *eukaryotic* cells the nucleus is bounded by a nuclear membrane and the genetic material is contained in the nucleus. In multicellular organisms eukaryotic cells are of two types—*somatic* (body) cells, and *reproductive* cells. Somatic cells divide by *mitosis* (asexual division) while reproductive cells divide by *meiosis*, or division to produce sex cells or *gametes* (the sperm or the ova). Thus mitosis helps in growth and development while meiosis produces gametes for sexual reproduction.

In mitosis the daughter cells each acquire the same number and kind of chromosomes as the mother cell. Hence, mitosis is given the name *equational* cell division. Mitosis ensures that all cells of an individual are genetically identical to each other and to the original fertilized egg. Such cell division is often clearly visible under a microscope.

Cell division of eukaryotic cells can also be by meiosis. Meiosis gives rise to four gametes from each mother cell dividing, each gamete getting *half* the chromosome number (hence the name *reduction* division) of the parent cell. An important feature of meiosis is that it splits the original cell's chromosome pair into *one* chromosome each in the gametes formed. Since the two original chromosomes (called *homologous* chromosomes) in the pair as present in the parent cell are not in general identical, meiosis ensures that the four gametes formed will not all have identical chromosomes. This produces genetic variability in the gametes. Subsequently, fertilization (the fusion of sperm and ova nuclei) brings together assorted and diverse single chromosomes, creating a zygote (a fertilized female gamete) with a full complement of two chromosomes, one coming from each parent. This results in both inheritance and genetic variability (noted to be a key requirement in evolution) in the progeny.

Owing to its significance, we repeat this explanation. In every sexually reproducing organism, chromosomes exist originally as a *pair* of two distinct chromosomes, each containing half of the organism's genetic information. Each human cell carries 23 such pairs. In each parent organism these collection of pairs are identical in each cell, all having resulted originally from the subdivision of a single complete (fertilized) cell. Following mitotic cell division, the daughter cells have the same chromosome complement as the parent cell. However, in meiosis, the number of chromosomes in the sex cells produced is half—each sex cell contains only *one* of the pair of the chromosomes in the parent cell. The process of meiosis—the splitting of each parent's chromosome pairs in the formation of sex cells—is a key step in the transfer of inheritance. During meiosis, in each parent two chromosomes line up in pairs and then pull apart, one member of each pair afterward moving to each of the sex cells being formed. Subsequently, the union of eggs and sperm during sexual reproduction restores the full amount of genetic information (by the formation of a *pair* of chromosomes, one coming from each parent) in the progeny being produced. Thus, during sexual reproduction, genes are transferred from parents to progeny, completing, according to the chromosomal theory, the transfer of inheritance.

The chromosomal theory of inheritance has now strong support of a large number of experiments using bacteria, fruit flies, plants and other organisms. To recall, this theory states that the units of heredity (genes), each determining a particular characteristic of the organism, are located on chromosomes (DNA strands). They later segregate and independently *assort* during meiosis. Sexual reproduction involves the fusion of two parental reproductive cells (ova and sperm, the gametes) in the process of fertilization. Any and all of the parents' characteristics that have to be genetically inherited must therefore get first put into the parents' genes. Such genetically acquired information (resident now in the progeny DNA) passes to an intermediate material (the *m*-RNA), which subsequently triggers the synthesis of appropriate proteins that constitute the body (and the consequent behavior) of the progeny organism. Proteins determine the phenotypic properties of an organism.

Based on the bio-chemical steps involved, however, the possibility of the *reverse* process (features of a protein passing back as information to DNA) seems extremely unlikely (Smith, 1993). Consequently, it is currently believed that *acquired* characters developing in an

organism's structural features cannot pass characteristics *backward* to DNA. Thus, information can pass only from DNA to protein. Further, as Smith notes, *even if* the protein to RNA to DNA information transfer process could occur (as has been reported with some animal viruses), a change in an organism's phenotype (manifested in the physical features or observable behavior of the organism) to be translated into a change in a protein is a phenomenon that is not yet supported by observations made in the natural world.

The above facts weaken support for Lamarckism as the explanation of biological evolution. Still, the story of evolution itself continues to be engaging, as follows.

7.2 EVOLUTION IS ALSO CULTURAL

Even though an organism may not be able to pass on its acquired characteristics to its progeny genetically, the organism may change its own *behavior* considerably during its lifetime—to adapt itself better in its own immediate environment. Recall that the term adaptation implies *any* process whereby a structure or behavior is progressively modified to give better performance in its environment. Indeed, observations in the natural world indicate that *habit formation* (the tendency to act constantly in certain manner) is also a part of adaptation. Additionally, it is seen that habits of a population change or evolve far more rapidly than its genetic make-up. Such evolution is *cultural*. Great and Blue Tits—species of birds found in urban England—for example, now routinely peck through the lids of milk bottles standing on door steps to get at the cream, the process having been discovered to be rewarding by some tit Prometheus, only sixty or so years ago. Thus, in addition to genetic inheritance, such behavior changes that we may call *learning* also assist the organism in its adaptation to the environment, but note that we have no evidence yet that learning is passed along genetically!

What role then does learning play in an organism's development and could this inspire a strategy for GAs beyond GAs emulating Darwinian evolution? Natural selection is a slow process and evolution occurring by natural selection and genetic inheritance alone can get "stuck" at some stage of evolution for a long period, if sources of variation are relatively few. This is confirmed by studies of

population genetics (see Aarts and Lenstra, 1997, page 139). As the contrast, Smith (1993) cites the example of "historical evolution," a term he reserves for social, scientific and technological advances that mankind has enjoyed in the immediate past—the 100 years alone. No hereditary mechanism or process has been involved here whereas the degree of changes achieved is remarkable. Therefore, a better *adaptive plan* would possibly be one that would combine genetically inherited adaptation with learning, as one *combining* Darwinian adaptation (occurring through genetic variation and natural selection) with, for instance, Lamarckian adaptation (transfer of acquired or learned characteristics). These ideas may be variously developed to improve the simple Darwinian GA described in Section 2.6.

Some notable other-than-genetic-inheritance contributors to evolution (what we call learning) are already exploited by living organisms as follows. Animals and plants can genetically adapt to changed conditions. But an additional ability of the higher forms of life is their *ability to learn* and thereby improve upon their genetically inherited performance. This ability encompasses characters such as temperaments, beliefs, prejudices, consciences and talents, which individuals can develop through individual conditioning. Such developed characteristics, however, belong only to those individuals who undergo learning.

Learning, nevertheless, is an amazing capability. By learning, individuals can develop a variety of different patterns of behavior in different circumstances. Such individual differences are not traceable to genetic differences, but rather to what is called *cultural* differences. Note, however, that in the *natural* world, culturally or socially acquired characteristics are *not* genetically transmitted to the offspring.

This points up to an opportunity and thus a possibility to hasten performance improvement through appropriately *engineered* evolution—if such "engineering" were feasible. However, natural evolution does not seem to aim directly at performance improvement or progress, rather it moves the population through a succession of *local adaptation* (see remarks by Dawkins in the preface of Smith, 1993).

In the artificial world of GAs the popular strategy, beginning with Holland (1975), has been to emulate the natural evolutionary process steps—mutation, crossover, inversion, niche formation, speciation,

etc.—that manipulate the *genetic* material, and then to subject the population periodically to selection to improve the solutions. Thus the conventional GA (what we call the Darwinian or "natural" GA) emulates only the natural or genetic evolutionary processes to speed up the search for the global optima. This seems to work fine in many situations. However, a key objection of those who have used GA to solve large and complex combinatorial or other problems is that Darwinian GA is slow; it often takes a long time to produce competitive solutions or solutions of respectable quality.

Notwithstanding natural evolution, a critical advantage we have in the artificial world is that in it we are frequently able to *force* acquired or learned characteristics back all the way to "alter the organism's genetic material," with no particular difficulty. As we saw in Sections 4.8 and 4.9, when this can be done, the results are remarkable when compared with the performance of the simple, "natural" GA. The key reason is that in the application domain of GA (such as flowshop sequencing by GA), a chromosome is a coded job sequence representation while its phenotype is the job schedule to be implemented in the factory. The genotype and the phenotype of a member of such GA population are thus identical. A change in the phenotype of an individual here therefore can be directly translated back into a change in its genotype. This overcomes the difficulty faced in the natural world in which even though a change in RNA may in some rare situations cause a change in DNA, the protein to RNA change (i.e., causing a protein change into a change in genotype) is not possible (Smith, 1993). Albeit such a stand deviates from the GA being a "pure emulation of nature," but such a stand may needlessly bind us from being effective explorers of global solutions (see Aarts and Lenstra, 1997, page 137).

In nature, learning is managed by the neural system of the organism. Some questions immediately pop up. Can we construct a "neural system" for the individuals in GA that can smartly manipulate information to achieve learning? Such a system by design would not wait for the slow process of adaptation to solve the problem (as done in Holland's GA), rather, it would attempt to improve the individual's performance *further* in "*this very*" generation. Some issues to be resolved here are as follows. What is it that we would call the ability to "intelligently manipulate" information? What would be "information" here? What would be the relevant "knowledge"? How would we encode knowledge to make it comprehensible to

"intelligence"? How would we structure information here and how would we store it for a particular chromosome? Perhaps an answer may come here from a richer GA (data-) structure.

Alternatively, can intelligence be applied *externally* so that the elegant structure of Holland's chromosome remains more or less intact? Note that such structure makes it straightforward to process the chromosome by mutation, crossover, etc. One possible idea is to take the best (or a few) GA solution(s) in a given generation and apply any and all known heuristic methods (Section 1.3) to improve its/their fitness or performance. Such heuristics may come from domain-specific knowledge available in the problem domain (such as shop scheduling or the travelling salesman problem, some listed in Section 4.5, for instance). If the solution improves, keep the improved version. Or else keep the original. Many variations to such schemes are possible. We may re-arrange the whole chromosome, or perhaps an appropriate part of it. Such may be the application of externally applied intelligence—to help GA solutions improve themselves within their own generation. We witnessed the construction and efficacy of one such method in Sections 4.8 and 4.9. There the incorporation of learning (by adding *solution-improvement* or local search heuristics in the GA loop) yielded significant improvement in the GA's convergence rate.

But the story of evolution does not end with one single population improving its adaptation, because the naturally evolved world is noted to consist of stable and coexisting populations of a "thousand" species, not just one. A *species* is a distinct group of individual organisms that resemble each other in most reproductive, physiological, biochemical and behavioral characters. These individuals are capable of breeding with each other under natural conditions. However, a distinct feature here is that members of two different species cannot interbreed sexually.

How did these different *species* form? You might now ask, if that question really relevant to problem solving by GA? We shall see in the next chapter that the answer to this question has led to the construction of more powerful GAs that can overcome *genetic drift* (the population converging to a single genotype) and help solve multi-modal and multiobjective optimization problems. To set the stage, we recall below the factors that are believed to have led to *speciation* or the formation of the multitude of species in nature.

7.3 THE NATURAL WORLD OF A THOUSAND SPECIES

One remarkable feature of the ecosystem is the coexistence of the relatively stable populations of tens of thousands of *types* or *species* of plants and animals. As we noted in Chapter 2, Darwin's book "On the Origin of Species" (Darwin, 1860) was the first to propose that new species of organisms formed not abruptly as the Book of Genesis states, but rather by a process that Darwin called natural selection. *Reproductive isolation*, he explained, along with natural selection, causes speciation.

To support his theory of natural selection, Darwin enlisted many basic facts. For instance, he pointed out that all organisms tend to overproduce. If there were no environmental checks, sparrows and fruit flies would increase rapidly in number. In spite of such high reproductive potential, however, the number of individuals in a species remains relatively constant, suggesting a struggle for existence. Also, there is the inevitable competition among organisms for the resources they require to survive—space, food, nesting sites, water, etc. Additionally, one observes genetic variations even within a single species. In the face of struggle for existence, individuals who have favorable genetic variations demonstrate a better frequency of living long enough to reproduce and pass their advantageous characters to the next generation. A well-documented case is that of the British *Biston betuaria* moth—a favorite prey for birds—which in 75 years evolved from light to dark coloration as industrial smoke darkened the tree trunks on which they lived (Smith, 1993; Russell, 1998).

For struggles facing an organism, Darwin noted three specific types. *Intraspecific* struggle (organisms competing with each other for scarce resources of common use) takes place between members of the same species. If this is severe it may result in fatal consequences as the needs of these individuals are similar. *Interspecific* struggle occurs between individuals of *different* species. In a community of different organisms, some organisms feed on others which frequently affects the "lower" animals—those with less sophistication do not survive. Organisms also must contend with adverse *environmental* conditions. Excess of moisture, draught, extreme heat or cold, etc. can make survival difficult.

Natural selection is a natural process that promotes the survival and reproduction of certain organisms that are carriers of some genetic endowments and inhibits others. A formal definition given by biologists today states natural selection to be the heritable differences in the ability of several *genotypes* (the hereditary characteristics of an organism) to produce viable offspring, and the changes in genotypic proportions that result from these differences (Magill, 1991). Darwin suggested that natural selection comes about due to the struggle for existence—between members of the same species or between individuals belonging to different species, particularly under unfavorable environmental conditions or when predators and parasites increase.

Around 1925, biologists also began probing carefully into the peculiar patterns of food web use by different organisms. Their studies hinted that considerable *interaction* exists among organisms sharing similar food. These studies concluded that relatively stable subpopulations of different species develop dependent on how organisms interact with one another, and with their environment. Subsequently, a "ecological theory" was proposed by investigators who closely tracked populations, made careful observations and then proposed a deductive mathematical theory. Evidence now exists that special interactions between organisms and their environment, and between the organisms themselves, have much to do with the variety manifested in the form of the multitude of species that we observe in today's coexisting life forms.

The "survival of the fittest" (a term coined by Herbert Spencer in 1910) principle is the principal contributor to the gradual change of organisms generation after generation, according to Darwin. The principle states that in face of competition and/or environmental affliction, organisms with advantageous variations in characteristics are more likely to survive. Gould (1996) notes that Darwin was inspired to think along these lines by the writings of Adam Smith, who proposed the doctrine of *laissez-faire* economics.

Somatic variations (strong legs developed by exercise, or a well-fed and healthy constitution) may enable a particular individual organism to survive many times during its lifetime. But, August Weismann working between 1889 and 1893 pointed out that somatic variations are never inherited by offspring; such variations die with the organism and have no effect on evolution (see Smith, 1993, pages 78-

81). However, if any advantageous characteristic is *genetic*, it may be passed on to the offspring generation after generation and then the *progeny organism* also becomes better suited for survival.

Genetic variations can occur within a species by recombination, or by mutation. *Heritable* variations are genetic variations, passed on within families and to the offspring from parents. It is through heritable variation and the survival-of-the-fittest operative, Darwin had suggested, and many subsequent observations support, that nature selects and evolves new organisms and even new species. Before we close this discussion, we must point out that *mutation* of genes is non-adaptive. Mutation is directionless and it does not cause adaptation except by accident (Smith, 1993). (Also see Russell, 1998, pages 617-618.)

Some of the special interactions between organisms and their environment, and between the organisms themselves, occurring due to the survival-of-the-fittest force, however, are quite complex and even today they are not completely understood. One complexity is phenotypic modification resulting from phenotypic reactions to environmental stimuli. These modifications are not hereditary. But these can reduce the selection pressure so that organisms gain some generations to produce favorable gene variations. Still, there are aspects of the behavior of the *overall* ecosystem that can be conceptualized and explained, and then used as paradigms for the construction of powerful problem solving algorithms. We touched upon some of aspects in Chapter 2. We expand these in the following pages.

7.4 KEY FACTORS AFFECTING THE FORMATION OF SPECIES

One important aspect of the modern theory of evolution is how it explains natural selection. In the modern view, natural selection is a consequence of *differential* reproductive performance (the differing number of descendants they produce) of different organisms. An organism's reproductive performance is determined by its genotype, as are its many other structural and behavioral characteristics.

The pressure of natural selection tends to conserve in a population features that are most advantageous or adaptive in the population's

environment. It is presumed today that genetically derived adaptations arise as a result of natural selection. However, broadly defined, adaptation is a shift in function or form, or both, that ensures fitness in a certain environment. Still, note that the term *adaptation* implies those changes (structural or in form) in an organism that evolve for the *current utility* of such changes. For a living organism adaptation supposes an adjustment of requirements and tolerances and the achievement of a certain efficiency so that it will grow better, reproduce more freely, and build up large populations. All plants and animals that exist and perpetuate themselves through several generations in any one environment or habitat must be adapted to that environment.

On the other hand, any particular *environment* can be characterized by certain distinct factors. The various types of factors that are active in an environment can be physical, chemical, and biotic. Physical factors include light, temperature, soil feature, etc. Chemical factors include the pH or the presence of dissolved gases in water, the presence of pollutants in water, air, soil, etc., and the presence of various mineral elements in soil, for example. Biotic factors include food supply and the presence and behavior of neighboring organisms—predators, flowers, bees, etc. Biotic factors in an environment can be symbiotic, in which each species present in that particular environment will contribute in some manner to the well being of others. Therefore, it is easy to visualize that the physical, chemical or biotic factors may differ considerably over different parts of the earth. This is one manner in which parts of a population diverge or get split from each other and thus adapt themselves to their *local* conditions can, over time, lead to the formation of distinct species.

Speciation, it is said, occurs when populations diverge so much that interbreeding can no longer occur between them. One important phenomenon that increases diversity of organisms in nature and to speciation is *niche formation*.

7.5 WHAT IS A NICHE?

Variation among habitats, as well as the drive for each species to survive and preserve their kind, leads organisms to specialize into particular roles they select to play in the ecosystem. The term *niche*

describes the *role* that a particular population (that is, the members of a single species) plays within the ecosystem. Factors such as eating habits, predatoriness, environment, etc. can determine such particular roles.

The *ecological niche* is the particular way an organism fits into the ecological system of which it forms a part. In this sense, a niche is the "position" of an organism within a ecological community. *Niche formation*, which occurs due to the effects that environmental factors have on organisms and different organisms have on each other, subsequently leads to the variety of distinct yet coexisting life forms. Specialization by an organism for a particular *task* in the environment helps stable subpopulations to form around these niches. Studies of overlap in resource use—food, habitat, humidity and pH, etc.—reveal interesting patterns of resource use and niche formation by the community to which the organism belongs. An organism's environmental niche is its range of tolerance to conditions (Figure 7.1).

A niche also portrays the range and the extent of resources (e.g. the availability of various sizes of insects for feeding) exploited by two different organisms along some resource continuum due to their mutual interaction (Figure 7.2). Two or more similar species will not be found inhabiting the same locality unless they differ in their food or in their breeding habits, in their predators or their diseases. For, if two species were identical in every respect, they would differ, however slightly, in their efficiency, and so the less efficient species would become extinct.

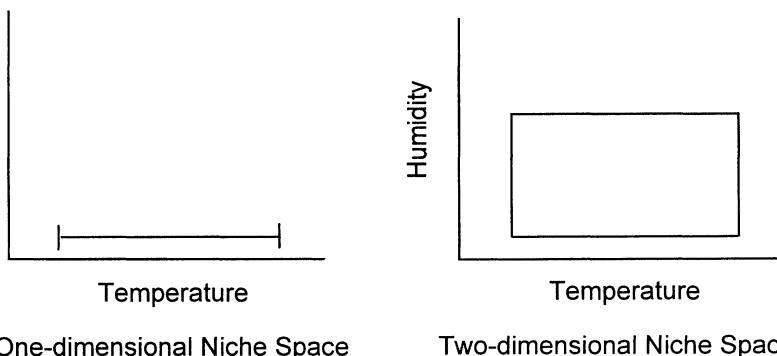


FIGURE 7.1 RANGE OF TOLERANCE FOR ENVIRONMENTAL VARIABLES

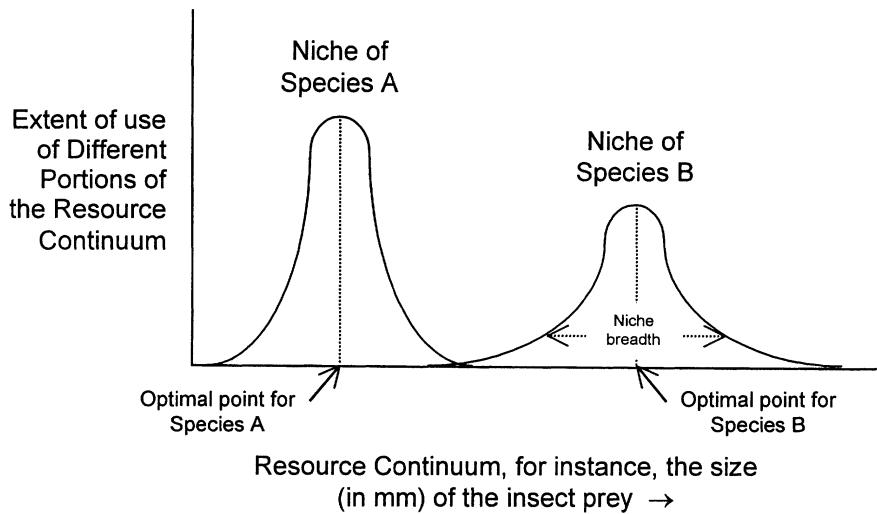


FIGURE 7.2 NICHES FOR TWO SPECIES SHARING ONE COMMON RESOURCE

Along any given resource axis the niche occupied by an organism is an *expressed* pattern of resource use. Such subdivision of an organism's role reduces inter-species competition for resources and promotes survival. Purely physical reasons (e.g. beak size) may determine the outer limits of the organism's "fundamental niche" within which it is capable of living and reproducing. A species will have an optimal point (e.g., insect prey size) on the resource axis where its performance is maximal.

7.6 POPULATION DIVERSIFICATION THROUGH NICHE COMPACTING

In practice, however, organisms rarely occupy their entire fundamental niche. Biotic interactions such as competition, predation and the predictability and availability of resources combine to shape the "realized niche" of an organism. This occurs because competitors and predators prevent it from being equally successful over the complete range of each environmental variable or resource. The realized niche thus is a subset of the fundamental niche of an organism.

Thus, competition, predation and predictability of resources affect the overall breadth of the niche, but each individual within a population of identical organisms is also exposed to continuous competition from its co-specifics. If competition with other species is not severe, intraspecific competition (competition within the members of the same species) will force individuals to try to exploit those parts of the potential resource range where competition with their fellows is reduced.

Thus, from an evolutionary standpoint, competition between members of the same species broadens niche space. Here individuals are forced to explore new food resources, microhabitats, etc. to the limits of their tolerance. Conversely, competition between different species (interspecific competition) narrows niche space for each species, since one species is likely to be more successful than another in exploiting the available resources.

While for an individual organism the effects of predation and of interspecific and intraspecific competition are the same, nature presses each individual to reduce its expressed niche, by encouraging each individual within the population to specialize in areas less fully exploited by its co-specifics. This promotes *diversification* of individuals within the population (members of the same species) and also a broadening of the pattern of resource utilization of the population (the members of the same species) as a whole. Still, no organism can afford to become too much of a specialist where conditions and the availability of resources may be unpredictable and changeable. If the organism insists on specializing here, it may not survive!

Nothing in ecology is clear-cut, however. Indeed, quite extensive overlap in resource use is seen to be quite common. Where resources are superabundant, substantial overlap may be tolerated. Several species of diving ducks wintering off the coast of Sweden coexist as do four distinct species of lake-dwelling flatworms. In both cases, when resources reduce, there is reduction in overlap. Alternatively, high overlap along one resource axis may be accommodated by a separation along some other dimension of the niche. Different earthworm-specializing birds may completely overlap in their diet (Figure 7.3), yet one feeds entirely in pasture while the other forages only in woodland. Additionally, several factors keep the species separate (Smith, 1993).

Niche shift (a change in the pattern of resource use) in the presence of competition occurs in the short term through behavioral modifications in resource use, and in the long term through evolutionary adaptation if shifts in resource use are maintained. The quantitative shifts in the size of termite prey taken by *typhlosaurus* skinks in the Kalahari desert is one of many such observed examples of niche shift.

Competition occurs when a number of organisms of the same or different species utilize common resources the supply of which is short, or if the resources are not in short supply, competition occurs when the organisms seeking these resources nevertheless harm each other in the process (Birch, 1957).

But, competition is not the only means that promotes specialization in nature. One phenomenon is symbiosis, exemplified by the African rhinoceros and the tickbird. The tickbird has evolved to live off ticks picked off the rhino while in the process the rhino gets rid of ticks. The Smyrna fig and a tiny wasp are entirely dependent on each other: The fig can produce neither fruit nor seeds unless this species of wasp enters the young flower and accomplishes cross-fertilization. The nitrogen-fixing bacterium *Rhizobus leguminosum* lives in the roots of the legumes. The bacteria are protected in the nodules which form on the roots and the legumes benefit by an increased supply of nitrates fixed by the bacteria. Perhaps many "non-symbiotic" pairs of organisms could not survive because they lacked symbiosis.

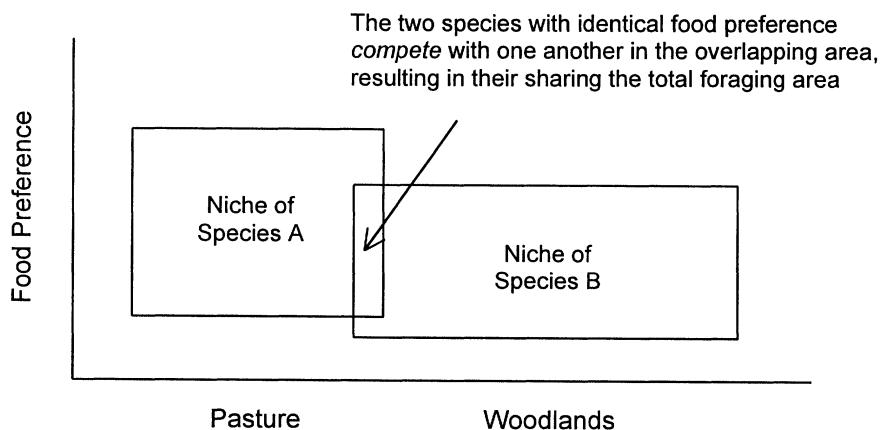


FIGURE 7.3 THE ECOLOGICAL NICHES OF TWO SPECIES COMPETING TO SHARE TWO DIFFERENT RESOURCES

Natural selection is also a means to differential reproduction. As described by Birch (1957), natural evolution may take place even when resources are not limiting, if the carriers of some genes possess greater reproductive potential than the carriers of other genes. This is how new characters get established in the population.

The additional role played by natural selection is the origination and evolution of *new* species, a process known as speciation. Recall that a species is a group of fertile organisms that can interbreed and produce fertile offspring only among themselves. Isolation and differential reproduction, as explained below cause speciation, a key object of Darwin's thoughts and studies.

7.7 SPECIATION: THE FORMATION OF NEW SPECIES

A *species*, as already mentioned, is a distinct group of individual organisms that resemble each other in most reproductive, physiological, biochemical and behavioral characters. These individuals are capable of breeding with each other under natural conditions. However, a distinct feature here is that members of two different species cannot interbreed sexually.

Recall that the term *population* in ecology describes a group of individuals that belong to a *single* species or of several closely related species that occupy a definite environmental area. The term *habitat* describes the particular environment in which a population lives.

The gradual formation of distinct species in nature is a process called *speciation*. Speciation simply means origin or evolution of *new* species. Populations that live in different localities or habitats may be exposed to different climates, foods, predators and parasites. Natural selection will then either enhance or diminish the frequency of different genes in different populations exposed to such different environments. A *race* is a population that differs in the frequencies of some genes or chromosomal forms from other populations of its species. Races of same species that inhabit different territories with different environments may in course of time (through adaptation) diverge genetically. Subsequently, a stage of genetic divergence may be reached when the genetic endowment of *each* race is a harmonious whole—a collection of certain genes and their alleles—that operate

best and produce a well-adapted organism when they are present together.

Gene exchange *between* such divergent races is liable to produce adaptively inferior, sometimes poorly viable or sterile recombination. It then becomes advantageous to the populations concerned to limit or even prevent altogether the gene exchange between them by reproductive isolating mechanisms and the "gene pool" is then said to become "closed." Note that *races* are *genetically open systems* and gene diffusion between them occurs frequently. *Species*, on the contrary, are *genetically closed*: gene exchange between them is absent or so limited that it is easily controlled by natural selection. Thus new species are formed.

A population is a geographically localized group of individuals of the same species. The total collection of all genes in a population is called a gene pool. Thus, the gene pool will have all the genes of the organism. Evolution is the event when the gene pool of a population changes. Such change can take place due to recombination, mutation or by natural selection. When genes are exchanged between populations, gene flow occurs. New genes may be introduced by immigration of individuals of the same species in local populations. When a population can receive new genes by interbreeding with members of some other populations of the same species, the gene pool is said to be open (Dobzhansky, 1970; 1986).

Thus, when populations are isolated by barriers—seasonal, geographic or otherwise, gene flow can be interrupted. Such interruptions lead to speciation when the gene pool becomes closed. The members of a population with a closed gene pool cannot sexually interbreed with individuals containing other genes. In such situations even if mating occurs, the progeny is sterile and can't reproduce its own kind. An example is the mule, a cross between the horse and the ass.

The development of speciation may then be described as follows. Consider the members of a single species constituting an initial, homogeneous population. Their genes are not all identical, but they have a common heritage and nature allows them to interbreed and exchange of genes occurs by recombination when offspring are formed. Next consider this population getting subdivided and each subpopulation gets partially isolated from the others. Each

subpopulation will then undergo its own evolutionary change through mutation and recombination, followed by natural selection, since the environments in which they live may be different.

One such subpopulation may get completely isolated from such other groups because of barriers such as a river or a mountain that its members cannot cross. Because of continued genetic divergence (through mutation and recombination) this isolated group will become phenotypically different from other subgroups. While this is going on, the other subgroups will also evolve, and they too can become phenotypically quite different subspecies or races. Note that the gene pool of races is still open and different races can interbreed. A good example is humans as a species consisting of African, White, Mongoloid, etc. races, developed as such in the last 50,000 years due to their ancestors getting isolated in remote locations on earth.

When after a long period the barrier between two separated races is removed, the two races will interbreed (this happens for human races) and form an intermediate race. This is the proof that the gene pools of the two races are still open. Alternatively, the different races may not be able to interbreed and hybrid races will not be formed. This will happen if the gene pools have become closed and the two races have become two distinct species. This is speciation or the process of the formation of new species.

Reproductive isolation is the fundamental test for new species formation. Reproductive isolation can occur because of differences in breeding seasons, courtship behavior, or other physiological reasons. A test that says that reproductive isolation has occurred is that even if offspring are produced, they are sterile, as is the mule.

To sum up, geographical isolation is regarded as the most essential condition for speciation. Without such isolation there will be no splitting of the gene pool and the formation of *stable* subpopulations that we call species.

In the next chapter we describe how even some such simpler aspects of niching and speciation have inspired powerful enhancement of the capability and efficiency of Holland's simple GA. In particular, we indicate how speciation can be induced in GAs to tackle multi-modal or even multiobjective optimization problems effectively.

Nature's ways, however, are far more intriguing and numerous than what has been captured by GAs so far. Even the simple GA does not fully exploit the simple process of mutation without considerable experimentation (selecting the "correct" probability of mutation for a given problem, an issue discussed in Chapter 3). As Smith (1993) notes, in experimental populations changes occur at a rate many order of magnitude greater than the rate characteristic of natural evolution (indeed we want this for the GA). But these changes continue for some time, and cease when all the available genetic variability has been fixed. No further progress occurs until new variability is generated by mutation. In evolution, and therefore also for the GA search to be effective, there must be a long run balance between the fixation of genetic variation by selection and the generation of new variation by mutation.

Dominance and diploidy, reordering, segregation, translocation, duplication and deletions are among some of nature's other implements. As time progresses, scientists and naturalists unwind and discover new mechanisms and natural processes almost nonstop. Many of these mechanisms have already shaped intelligent and powerful versions of genetic algorithms (see Goldberg, 1989, Chapter 5; Muhlenbein, 1997). In future we expect GAs to become better endowed and more efficient, though perhaps diverging from their purely "natural inheritance." One key reason is that nature itself *does not* seem to be aiming at achieving perfection in any sense (Smith, 1993). Another is that the design of such algorithms should be inspired by nature but there is no necessity here to copy nature exactly and stay in that mode. One such divergence is the incorporation of *Lamarckism* (a mode of evolution that today's biologists do not believe to be viable in nature) into GAs to hybridize them as outlined in Sections 2.8 and 4.6. This was inspired by the fact that humans are able to communicate their experience by speech, and later by writing, to their fellows and to subsequent generations who learn and thus improve.

Yet to be exploited to improve the GA are, therefore, a deeper understanding of population genetics through mathematical modeling (see, for instance, Muhlenbein's article in Aarts and Lenstra, 1997), and the finer manipulation of genes, learning, and the methods being inspired by the development of biotechnology (Russell, 1998). The object would be to make the GA both more effective and more efficient.

It is tempting to pose the question, can the capacity to learn be inherited? Irrespective of what the answer might be for organisms in the natural world, GAs can be designed to enhance and then exploit such capacity.

THE NONDOMINATED SORTING GENETIC ALGORITHM: NSGA

Unlike many search methods that develop a single "current best" solution and then try to improve it, a GA maintains a set of possible solutions called the *population*. At the intuitive level this would suggest that a suitably designed GA might be able to capture the members of the Pareto optimal set of solutions, if Pareto optimality were somehow used as the basis for measuring fitness. In this chapter we describe several approaches to endow GAs with the ability to capture and preserve the Pareto solutions in multiobjective optimization. We then describe the Nondominated Sorting Genetic Algorithm (NSGA), a multiobjective GA designed by Srinivas and Deb (1995) that seeks out Pareto solutions efficiently. A numerical problem, a bi-criteria robust design of an electronic filter, is then solved to illustrate its efficacy.

8.1 GENETIC DRIFT: A CHARACTERISTIC FEATURE OF SGA

It is commonly observed that a properly parameterized simple GA (Section 2.4) converges consistently to the global peak. Indeed this is a very *desirable* feature of SGA. However, if the function being optimized has multiple peaks all of equal heights, SGA still converges to a single peak due to a phenomenon called "genetic drift" (Goldberg and Richardson, 1987). Genetic drift is noted in nature also when certain conditions prevail (Russell, 1998). As we noted in Chapter 6, however, a Pareto optimal or efficient front is actually a set of solutions, not a single point, i.e. a *multitude* of nondominant members populates it. Consequently, any procedure devised to discover Pareto optimal (i.e., efficient) solutions should not delete, discard or destroy

solutions that are legitimate members of the front. Rather, the procedure should ideally avoid genetic drift-type phenomena and safeguard all potentially efficient solutions found along the way.

In the natural world coexistence or diversity is made possible by niching (members interactively "agreeing" to *share* the available limited resources among themselves) so that they do not compete head to head to destroy "some quite like themselves" (Section 7.7). Nature also forms stable subpopulations of organisms by speciation through long periods of isolation or differential reproduction, thus preventing interbreeding. These observations have inspired the construction of several special GAs that exploit niche formation and speciation concepts to rapidly discover Pareto optimal solutions.

The first GA proposed for finding multiple solutions to multiobjective problems was VEGA (Vector Evaluated Genetic Algorithms), created by Schaffer (1984). Briefly reviewed in the next section, VEGA, however, does not use niching. Rather, it seeks its goal by selecting a fraction of the next generation by using each of the multiple attributes in consideration (i.e., cost, reliability, weight, etc.) turn by turn. But VEGA never selects solutions according to tradeoffs among these attributes (i.e., those on the "middle" area of the Pareto front); hence, frequently it finds only the extreme points on the Pareto front. About the time when VEGA was proposed, Deb and Goldberg (1989) evaluated the effectiveness of niche formation and speciation in genetic function optimization, specifically for the difficult-to-optimize multi-modal functions. This work inspired the construction of several other procedures.

A niche-based scheme to find Pareto optimal solutions was proposed by Horn, Nafpliotis and Goldberg (1994). These researchers utilized niching pressure (by *sharing* fitness) to spread the population out along the Pareto optimal surface. Another approach has been developed by Srinivas and Deb (1995) who use niching, as well as *nondominated sorting* of the solutions in every generation to ensure that the "good" solutions (those on or near the Pareto front or those which dominate other points) get preference in selection for procreation. This new approach is known as NSGA (Nondominated Sorting Genetic Algorithm).

The suggestion to perform nondominated sorting came originally from Goldberg (1989) who saw this as one good way to give equal

reproductive chance (the likelihood of getting selected for mating) to *all* solutions that are at the same level in the Pareto-dominance sense.

All the above studies are well documented and worth reading for anyone interested in developing or applying evolutionary heuristic methods for multiobjective problems. In the following sections we provide brief summaries of these studies and a numerical illustration of applying NSGA.

8.2 THE VECTOR EVALUATED GENETIC ALGORITHM (VEGA)

The original idea of using genetic search in multiobjective problems is traceable to Rosenberg's (Rosenberg, 1967) suggestions for handling multiple properties to solve such problems. Subsequently, Schaffer (1984) proposed an approach that he called vector evaluated genetic algorithm (VEGA) to solve such problems. Schaffer modified Grefenstette's GENESIS program (Grefenstette, 1983) to include multi-criteria functions in it. His approach creates equal-sized subpopulations for selection along each of the criteria components in the evaluation vector. The key feature of VEGA is that the selection is performed *independently for each criterion*, though mating and crossover are performed across subpopulation boundaries (Ritzel, Eheart and Ranjithan, 1994). Schaffer created a loop around the traditional procedure so that selection is repeated for each criterion or objective to fill up a portion of the mating pool. Then the entire population is thoroughly shuffled to apply crossover and mutation operations. This is done in order to achieve mating among individuals belonging to the different subpopulation groups, the idea being that of combining "building blocks" (Section 2.9.1) that lead to good performance along some particular objectives or criteria.

A problem with this scheme (as pointed out by Schaffer himself) is that the selection procedure here is biased against members in the "middle" i.e. solutions that are not excellent along any criterion. In VEGA survival or selection pressure is applied favoring extreme performance on at least one criterion.

Under the above scheme, if a utopian individual (i.e. one who excels on all criteria of performance) exists, then it may be formed by the genetic combination of some features of extreme parents. But for

many multi-criteria problems, this Utopian solution doesn't exist. For these problems, the location of the Pareto-optimal set or front is the accepted rationale. This front will contain some members with extreme performance on each different criterion and some with "mudding" performance on all criteria. These *compromise* solutions are of utmost interest, but there remains the danger of these solutions not surviving VEGA's selection process. This danger is usually more severe for problems with a concave Pareto optimal front, than for those with a convex one (Schaffer, 1984). To overcome this difficulty with VEGA, Schaffer introduced several ideas including a re-distribution scheme and a crossbreeding plan, though none appear to be very effective.

From the Pareto optimality standpoint (see Figure 6.1), any bias against the "middle" solution on the Pareto front is not at all desirable. These solutions are all also locally nondominant and hence carry appropriate building blocks. Therefore, such solutions are as attractive as those near one or the other extreme ends of the Pareto front. In fact, as Goldberg (1989) points out, if we accept the rationale of Pareto optimality, *all* of these individuals should participate equally in the creation of progeny through mating. We return to this important point in Section 8.4.

8.3 NICHE, SPECIES, SHARING AND FUNCTION OPTIMIZATION

We described in Chapter 7 how *sharing* of available resources has played a key role in the creation of the remarkable degree of diversity (multitude of species) found in nature. If sharing does not take place, the strongest would dominate and annihilate the weak. In nature such sharing occurs when different organisms, through interaction with each other, arrange to specialize in certain parts of the total span of resources available and form niches (particular tasks or roles), be it for habitat, food, or something else. In artificial genetic search also, we see that sharing through niche formation in a population of string chromosomes leads to very useful outcomes. This section briefly recalls the findings of a key research on the merits of artificial niche formation and speciation.

Deb and Goldberg (1989) were motivated to find a way to avoid "genetic drift," the phenomenon in which the simple GA converges toward a single peak (the global optima) of a multi-modal function (a function with multiple peaks) being optimized. Genetic drift occurs even if there are several peaks of *equal* size present. They investigated two processes, namely niche formation by sharing, and speciation induced by imposing mating restriction. They noted that sharing would help maintain subpopulations at multiple peaks, while speciation would improve the overall performance of the algorithm by restricting mating of members and the consequent increase in the exchange/formation of good building blocks *within* the members of each niche.

Deb and Goldberg considered two niching methods. One is called crowding while the other is sharing. Crowding, suggested by De Jong (1975), creates separate niches by replacing existing members in a niche according to their similarity with other members in an overlapping population. Sharing is the other approach used to form niches and it occurs in nature when certain resources are shared within an environmental recess. In its GA analog, the "resource" shared is a member's fitness, a metric used in GA to base selection (hence the member's survival) for mating. To induce sharing in GA (a concept attributed to Holland (1975)) a member's fitness is derated by an amount proportional to the nearness of other members in its neighborhood. This reduces the tendency of a "high fitness" single member to dominate the other members belonging to the same niche during selection.

One sharing scheme, tested by Deb and Goldberg, was proposed originally by Goldberg and Richardson (1987). In this scheme the population is divided into subpopulations according to the similarity of the individuals. "Similarity" is controlled by comparing the distance ($d(x_i, x_j)$) of individual members from each other, to a parameter called σ_{share} . How much sharing will occur between two members is determined by a power law sharing function $Sh(d(x_i, x_j))$, a function of $d(x_i, x_j)$ (the distance defined for two individuals i and j), as

$$Sh(d(x_i, x_j)) = 1 - [d(x_i, x_j)/\sigma_{\text{share}}]^\alpha \text{ if } d(x_i, x_j) < \sigma_{\text{share}}$$

$$= 0 \text{ otherwise.}$$

when α is the power exponent, suitably chosen. Fitness (definition based on the problem being solved) is treated as the "resource." To promote niching behavior, individual members within each niche are made to "share" their fitness with their neighbors.

The sharing function $Sh(d(x_i, x_j))$ determines what is a neighborhood and also the degree of sharing that each member will endure. σ_{share} controls the span of each neighborhood.

For a given member x_i , the degree of sharing (the reduction or degradation of its fitness which eventually controls the member's chance of getting selected for mating) is determined by summing the sharing function values contributed by all other members in the population to x_i . As Goldberg and Richardson indicate, members close to x_i will force a high degree of sharing while those far away from it will cause a very small degree of sharing.

The actual "after sharing" or derated fitness $f_s(x_i)$ of the individual x_i (to induce niching behavior among the members of the same niche) is calculated by taking the unshared original fitness $f(x_i)$ of the member and dividing it by the sum of sharing functions of that member (i) with each member (j), $j = 1$ to n in the population of size n , i.e.,

$$f_s(x_i) = f(x_i) / \sum Sh(d(x_i, x_j)) \quad (j \text{ is summed from 1 to } n.)$$

Thus, when there are many individuals in the same neighborhood, they contribute to each other's sharing functions, thereby bringing down one another's fitness.

The above mechanism simulates sharing of limited resources by organisms that form a niche in the ecosystem. Niching causes resources to be shared and thus it limits the uncontrolled growth (through repeated probable selection) of individuals with "high" fitness.

To recap the central idea, a niche is an organism's particular task or behavior in the natural environment and a species is a collection of organisms with similar behavior. The subdivision of the environment, holding the resources of value to the organisms, on the basis of the organism's specialization (around its chosen niche) reduces inter-species competition for these resources. This reduction in competition

helps *diversity*, i.e., *stable* subpopulations form around different niches in the environment. Further, *within* each niche, sharing of resources occurs to permit the members within the niche to coexist.

In the scheme described above, the parameter σ_{share} is of utmost importance. It controls the width of the neighborhood and the maximum distance between members who will share fitness and will thus be induced to form a niche. In a multi-modal optimization problem, the interest is in finding solutions corresponding to each mode (or peak) of the function being optimized. Therefore, one should choose σ_{share} to form as many niches as there are peaks (for functions with unknown number of peaks this may require some experimentation with different values of σ_{share}).

The distance $d(x_i, x_j)$ between the members may be evaluated either in the decoded parameter space (the space of the real decision variables), or in the mapped gene space. When distance is measured in the decision variables space, sharing is termed phenotype. When $d(x_i, x_j)$ is defined in the gene space, sharing is genotype. One finding of Deb and Goldberg (1989) is that phenotype sharing leads to better performance in the discovery of peaks of a multi-modal mathematical function.

Adding a mating restriction to promote speciation was another aspect investigated by Deb and Goldberg. They found that speciation improves on-line performance (rapid convergence of the GA to acceptable solutions) of GA search, in the following manner. Once the sharing scheme clusters subpopulations of solutions at the peaks of a multi-modal function, when recombination between members sitting on different peaks occurs it may produce new members that do not belong to any peak. Such recombinations are "lethal" for these often produce solutions of lower fitness that do not survive selection. Thus such recombinations waste computing effort and degrade the on-line performance of the GA.

It is therefore reasoned that recombination (crossover) between individuals belonging to different peaks should be avoided or at least minimized. One way to do this is to *induce* speciation, i.e., restricting mating between members belonging to different species. Nature does this by creating separate species at each niche and by restricting successful mating between different species.

The manner in which Deb and Goldberg induced speciation was to almost re-do what they did to induce niche formation. They used here a parameter that they called σ_{mating} . As in niche forming, they created subpopulations the members of which were allowed to mate (i.e., these members constituted a species) based again on a $d(x_i, x_j)$ -type distance-metric evaluated in the decoded decision variables space. In this way they imposed a phenotype mating restriction: individuals were allowed to mate when they were within a distance bounded by the limit σ_{mating} . In short, Deb and Goldberg implemented mating restriction or speciation control by applying the following rule: To find the mating companion of an individual, if another individual within a distance of σ_{mating} is found, mating is performed. Otherwise, another individual is tried. If no such individual can be located in the neighborhood of the first individual, a random individual is chosen to mate with the first individual.

Subsequently, using computer simulations Deb and Goldberg showed that mating restriction (induced speciation) indeed improved on-line performance of the GA when the object was to rapidly populate the peaks of a multi-modal function. They also noted that sharing done in the phenotype domain (the space of the real decision variable produced better on-line performance than when sharing and/or speciation was done in the genotype domain (the domain of the *coded* variables with which the GA works).

In summary, this study showed the improved value of inducing niche formation not by crowding, but rather by *sharing* (derating fitness by an amount proportional to how many "similar" members are in the immediate neighborhood). The study also demonstrated the utility of inducing speciation by restricting mating between members belonging to different peaks. Niche formation *spreads* the solutions to a multi-modal function to different peaks, thus it minimizes the chance of all solutions converging on a single peak. Speciation minimizes the lethal formation of inexpedient members (these have poor fitness) caused by undesirable mating, thus it improves the *rate* at which the GA finds acceptable solutions.

Later we shall see that these two processes emulated from nature constitute two vital aspects of GA search in the rapid and effective discovery of Pareto optimal solutions to multiobjective optimization problems.

8.4 MULTIOBJECTIVE OPTIMIZATION GENETIC ALGORITHM (MOGA)

The Multiobjective Optimization Genetic Algorithm (MOGA) developed by Fonseca and Fleming (1993) is also an approach for finding Pareto optimal solutions. It uses a solution ranking procedure to suitably position the different solutions relative to each other, in order to rank their chance of selection. In MOGA, the whole population is sorted for nondominance and all nondominated individuals are assigned rank 1. The dominated individuals are subsequently ranked by checking nondominance of them with respect to the rest of the population, as follows:

For an individual solution, the *number of solutions* (n) that strictly dominate the solution in the population is first found. Thereafter, a rank is assigned to that individual, equal to $n+1$. Therefore, at the end of this ranking procedure, there could be a number of individuals having the same rank. Clearly, the lower the rank value, the *more dominating* a solution will be. The selection procedure then uses these ranks to select or delete blocks of solutions to form the mating pool. However, it has been noted that this type of blocked fitness assignment sometimes produces a large selection pressure leading to premature convergence (Davis, 1991).

MOGA also uses a niche-formation method to distribute the population over the Pareto-optimal region. But instead of performing sharing on the genotype or phenotype parameter values, MOGA uses objective function values for sharing. Hence, even though this maintains diversity in the objective function values, this approach may not maintain diversity in the solution set, a matter usually of importance for a decision-maker. Moreover, MOGA would not find multiple solutions in problems in which different Pareto-optimal solutions correspond to the same objective function value.

8.5 PARETO DOMINATION TOURNAMENTS

Yet another approach to seek Pareto optimal individuals is given by Horn, Nafpliotis, and Goldberg (1994). Their method, called the Niched Pareto Genetic Algorithm, uses *Pareto domination tournaments* to guide selection for mating instead of a nondominated sorting and ranking selection procedure. Briefly, in this method, a *comparison set*

comprising a specific number (t_{dom}) of individuals is picked at random from the population at the beginning of each selection process. Then two individuals are picked randomly from the population and subjected to a tournament selection (for selecting a winner) in accordance with the following procedure. Both individuals are compared with the members of the comparison set for domination with respect to the *objective functions*. If one individual is nondominated and the other is dominated, then the nondominated individual is selected. On the other hand, if both are either nondominated, a niche count is calculated by simply counting the number of points in the population within a certain distance (σ_{share}) from that individual. The individual with the lower niche count is selected. The effect of multiple objectives is accounted for in the nondomination calculation.

Since nondomination is found in this procedure by comparing an individual with a randomly chosen population size t_{dom} , the success of this algorithm is highly dependent on the choice of the parameter t_{dom} . If the choice is improper, true nondominated (Pareto optimal) points may not be found. If t_{dom} is too small, this may result in a few nondominated points in the population. On the other hand if t_{dom} is too large, premature convergence may result. σ_{share} controls the extent of niching pressure that forces fitness sharing. The researchers demonstrated the utility of this method using a bi-criteria numerical optimization problem earlier solved by VEGA. The tournament method produced better distributions on the Pareto front.

The researchers also state that it is important to have a large enough population to conduct the GA search effectively and to sample the breadth of the Pareto front. Further details of this method may be found in the original paper by Horn, Nafpliotis and Goldberg (1994).

8.6 A MULTIOBJECTIVE GA BASED ON THE WEIGHTED SUM

A recently published method (Murata, Ishibuchi and Tanaka, 1996) to produce Pareto optimal solutions uses a weighted sum of the objective functions being maximized. The weights are variable, aimed at controlling the selection of parents to breed the next generation. The method also uses an elite preserve strategy.

The investigators have used the method to develop multiobjective Pareto optimal solutions for flowshops. They have also compared their proposed method to VEGA (Schaffer, 1984) and claim that the proposed method performs better than VEGA.

8.7 THE NONDOMINATED SORTING GENETIC ALGORITHM (NSGA)

The idea of nondominated sorting originated from Goldberg (1989), who noted that the problem of selection bias against the "middling" points (solutions) on the Pareto front is a serious one. If the rationale of Pareto optimality is accepted (see Figure 6.1), then the middle points should be accorded the same *reproductive potential* (Darwin's measure of fitness (Russell, 1998)) as those near the extremes. The reproductive potential of a solution is its probability of getting selected to mate in order to propagate its genetic material to progeny. Goldberg suggested the use of nondominated sorting to achieve this end. Specifically, he said that the population should be ranked on the basis of nondomination. For this all nondominated individuals in the current population are to be identified and flagged. These individuals are to be placed at the top of the list and assigned a rank of 1. These points are then removed from contention and the next set of nondominated individuals is identified and assigned rank 2. This process is to be continued until the entire population is assigned a rank. To maintain appropriate diversity, selection based on nondomination should be used in conjunction with the techniques of niche formation and speciation—to assure formation of stable populations along the Pareto optimal front. Then, as the GA executes, stable, multiple subpopulations will arise along the Pareto front.

Perhaps the most effective implementation of Goldberg's suggestion is the creation of the Nondominated Sorting Genetic Algorithm (NSGA) by Srinivas and Deb (1995). NSGA differs from SGA in the manner the *selection* operator works. The crossover and mutation operators work as they do in SGA. However, before selection is performed, the population is *ranked* on the basis of the nondominated sorting concept, to emphasize Pareto-optimality. In addition to adopting the nondominated concept, Srinivas and Deb cleverly devised a special fitness metric, which they called "dummy fitness." This fitness became the surrogate fitness basis for selection in NSGA.

8.7.1 How NSGA Works

The procedure to discover Pareto-optimal solutions devised by Srinivas and Deb works as follows. An initial set of randomly generated solutions starts the process. The nondominated individuals among these solutions are first identified from the initial population of solutions. The initial nondominated individuals constitute the *first nondominated* front in the population and are assigned a *dummy fitness* value equal to the population size (i.e., dummy fitness = N). This dummy fitness is intended to give an equal reproductive chance to all the initial *nondominated* individuals. The object of maintaining diversity among the solutions is achieved by causing *sharing* of the initially assigned dummy fitness among these individuals (the method is described below in Section 8.7.3).

Next, the individuals in the first front are ignored temporarily and the rest of the population is attended to, as suggested by Goldberg. The procedure is as follows. Individuals on the *second* front are identified. These second front solutions are then assigned a new fitness value, which is kept *smaller* than the *shared* dummy fitness value of the solutions on the first front. This is done to differentiate between the expectedly superior members of the first front and the members of the second front. Then sharing is again done within the second front and the process goes on till the whole population has been evaluated and classified into successive fronts. This process leads to the creation of several successive fronts of "nondominated" individuals with each individual owning an appropriate shared dummy fitness.

Next, individuals in the whole population are reproduced according to their relative (shared dummy) fitness value. This approach facilitates the selection to be biased toward the nondominated regions of the Pareto-optimal front. The process results in the quick convergence of the population towards the nondominated region while sharing helps to distribute the individuals over the entire nondominated region.

Figure 8.1 shows the key steps in NSGA. The two key differences between NSGA and SGA are (1) reproduction according to dummy fitness in NSGA vs. according to real fitness in SGA, and (2) sorting of solutions in NSGA based on nondomination. Both maximization and minimization problems may be solved by this algorithm by suitably altering the selection process.

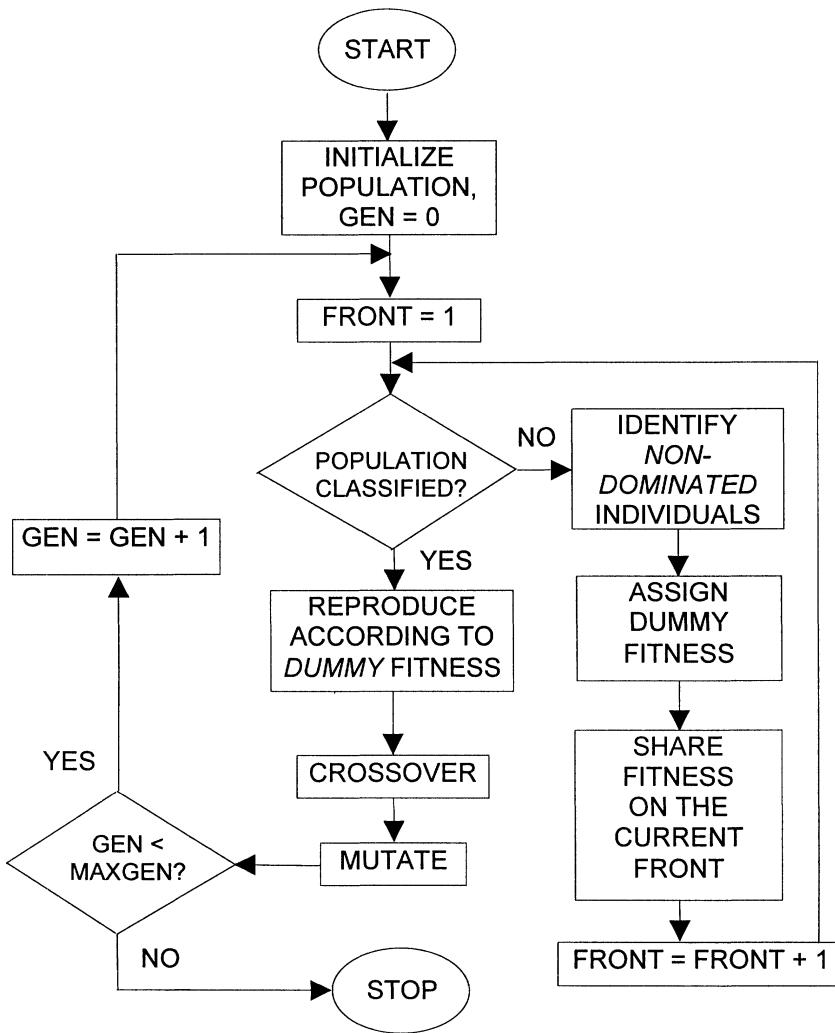


FIGURE 8.1 THE NONDominated SORTING GA

The efficiency of NSGA lies in the manner in which it reduces multiple objectives to be optimized to a single surrogate (dummy fitness) function using the nondominated sorting procedure. If a solution is locally dominated, it is globally dominated. However, the converse is not true (i.e., a solution's being locally nondominated does not imply its being globally nondominated) (Srinivas and Deb, 1995). In each generation NSGA finds locally nondominated points only.

However, if there exists any globally nondominated member in a given generation, NSGA increases the likelihood of its survival. This is because in the NSGA scheme, any such individual will have the highest possible dummy fitness.

Thus the power of NSGA lies in the successful transformation of a multiobjective problem, no matter how many objectives there are, to a single function problem, without losing the object of "vector" optimization. By selecting nondominated points, NSGA actually processes and propagates the schemata that represent Pareto-optimal regions. Therefore, the building blocks (in the sense Goldberg (1989) uses this phrase) for NSGA will be those schemata that represent characteristics of globally nondominated individuals.

8.7.2 The Concept of Fitness Sharing

The simple GA (SGA) has been criticized for its occasional premature convergence when substantial fixation occurs at almost all bit positions (i.e., many bits on the solution chromosome get fixed and do not change further even if the GA continues to execute for more generations) before the algorithm obtains sufficiently near-optimal points. But more seriously, the SGA is justly criticized for its sub-par performance on *multi-modal* functions.

To overcome this second shortcoming of the SGA, a method by which nature forms stable subpopulations of coexisting niches has been borrowed and capitalized on. The use here of *sharing functions*, a method by which the fitness of similar solutions are lowered, helps mitigate unbridled head-to-head competition between widely disparate points in the search space. Such fierce competition between the members of the same species is something rarely seen in nature (Smith, 1993). Rather, in nature, sharing helps maintain a diverse population stabilized in several niches. This leads to a more "considered" convergence of the population into coexisting subpopulations.

When the SGA operates unimpeded on a multi-modal function, it clusters all its points about one peak or the other. This is due to the SGA's design, which is based on the assumption made in the fundamental theorem of GA (the schema theorem, Section 2.9.3) of the

GA's operating with a population of infinitely large size. In a finite size population when no advantage exists for either of the two competing alternatives (peaks), the population will converge to one or the other of many in finite time. As already noted, this phenomena is called *genetic drift* and is undesirable in multiobjective optimization when no utopian solution exists (Section 8.2). In such situations we would be interested in determining a set of good solutions along with the "best" solution rather than a best solution alone. Thus the problem of determining the Pareto optimal front is similar to working with multi-modal peaks of unequal size on the fitness landscape. In seeking Pareto optimality also, it would be desirable to see a form of convergence that permits a stable subpopulation of individuals to cluster around *each peak* according to peak fitness.

In order to overcome the "clustering at the best solution" deficiency of SGA, the concept of niche (and species) has been borrowed from nature where different species don't always compete head-to-head. Instead they exploit separate sets of environmental features (niches) in which another organism has no interest.

Other strategies to maintain diversification to handle multi-modal problems also exist. Cavicchio in his Ph. D. dissertation introduced a mechanism called "preselection" (Cavicchio, 1970). In this scheme, an offspring replaces the inferior parent if the offspring's fitness exceeds that of the inferior point. In this manner diversity is maintained in the population because strings tend to only replace strings similar and inferior to them. Using preselection, Cavicchio was able to overcome the "clustering at the best solution" deficiency of SGA with a relatively small population size ($n = 20$). De Jong (1975) proposed a "crowding" scheme as an alternative. In De Jong's crowding individuals replace existing strings according to their similarity with other strings in an overlapping population. De Jong demonstrated good success with the crowding scheme in multi-modal functions. As we mentioned in Section 8.1, Schaffer (1984) used separate, fixed-size subpopulations in his creation of the VEGA. This is reported to work well with a number of trial functions, however, Schaffer himself expressed concern about the procedure's ability to fish out nondominated individuals in the "middle" that may be Pareto optimal but not extremal along any dimension.

In NSGA the initially assigned dummy fitness value of an individual gets reduced in proportion to the number of individuals in the

particular niche. So, in an area which has a very high density of individuals, those individuals' dummy fitness get reduced more, as a result of which their probability of mating gets reduced though the overall mating probability of a niche as a whole remains the same. So here instead of the individual getting selected, the objective shifts to selection from a particular set (i.e., selection of *any* member of a given niche) of individuals. All individuals in the niche have almost the same property, and therefore, if sharing is not done, then there is a high chance that in the total mating pool there would be more individuals with the same characteristics (leading to genetic drift). Logically, therefore, it would appear that sharing of dummy fitness is very essential for keeping diversity in the population.

8.7.3 Fitness Sharing in NSGA

In order to induce species formation in the population, intraspecies (i.e., "within same species") fitness sharing is necessary. For this two questions need to be answered. These are, (1) who should share the fitness, and (2) how much fitness should be shared? In nature these two questions are answered implicitly through conflict for finite resources (Smith, 1993). Diverse species seek out and specialize in different combinations of environmental factors which are relatively uninteresting to (or beyond reach of) other species. Such behavior is *niche formation*.

It might be possible to induce similar conflict of resources in genetic optimization, but unfortunately, in many optimization problems there is no natural definition of "resource." A *sharing function* is a way of imposing niche and speciation on chromosomes or strings based on some measure of their *distance* from each other. This enables the development of many solutions near a given peak (Srinivas and Deb, 1995). The sharing function is used to determine the degradation of an individual's payoff or fitness due to a neighbor at some distance as measured in some "similarity space" (see Deb and Goldberg, 1989).

As said in Section 8.3, a distance d_{ij} may be defined in two possible ways. The *phenotypic distance* between two individuals is measured based on the difference in the decoded (i.e., real-valued) problem variables while their *genotypic distance* is measured based on the difference in the coded problem variables (in the gene space) between

those two individuals. Also, in NSGA a sharing function $sh()$ is defined as a function of the distance metric d . Once d and the sharing function $Sh(d)$ have been selected, it is a simple matter to determine a string's *shared fitness* f' (to guide its selection in the GA sense) as its potential fitness f_i divided by its niche count, m_i .

$$f'_i = f_i/m_i$$

In NSGA fitness f_i is dummy fitness (Section 8.7.1). The niche count m_i is taken as a sum of all $Sh(d_{ij})$ of solution i taken over the entire population N . Thus

$$m_i = \sum_{j=1}^N Sh(d_{ij}) = \sum_{j=1}^N Sh(d(x_i, x_j)).$$

Since the sums include the string itself, if a string (# i) is the sole member of its own niche (i.e., for such a string $m_i = 1$), it receives as its shared fitness its full potential fitness. Otherwise, the sharing function degrades its fitness according to the number of points within its niche in its neighborhood.

Rintala (1998) proposes another variation of niche forming genetic algorithms known as the "parallel (diffusion) GA." This method uses information exchange between solutions only by crossover to induce niche formation. Local comparison is used to determine a solution's Pareto rank.

8.8 APPLYING NSGA: A NUMERICAL EXAMPLE

Multi-criteria decision problems are numerous not only in business and public administration, but also in technology development and engineering. A particularly rich assortment of multi-criteria problems exists in engineering design. In this section we solve a problem that comes from the domain of quality engineering, a recently formalized discipline that aims at developing products whose superior performance delights the discriminating user—not only when the package is opened, but also throughout their lifetime of use. The quality of such products is robust, i.e., it remains unaffected by the

deleterious impact of environmental or other factors often beyond the users' control.

Since the topic of quality engineering is of notably broad appeal, we include below a brief review of the associated rationale and methods. We then present the NSGA solution to a bi-criteria robust design problem. If you are familiar with the basic concepts of robust design, you may skip reading the next few pages and go directly to Section 8.8.3.

The term "quality engineering" (QE) was used till recently by Japanese quality experts only. One such expert is Genichi Taguchi (1986) who reasoned that even the best available manufacturing technology was by itself no assurance that the final product would actually function in the hands of its user as desired. To achieve this Taguchi suggested the designer must "engineer" quality into the product, just as he/she specifies the product's physical dimensions to make the dimensions of the final product correct.

QE requires systematic experimentation with carefully developed prototypes whose performance is tested in actual or simulated field conditions. The object is to discover the optimum set-point values of the different design parameters, to ensure that the final product would perform as expected *consistently* when in actual use. A product designed by QE has *robust* performance. We explain these steps in the next section to reduce QE to a multiobjective optimization problem.

8.8.1 The Secret of Creating a Robust Design

A practice common in traditional engineering design is *sensitivity analysis*. For instance, in traditional electronic circuit design, as well as the development of performance design equations, sensitivity analysis of the circuit developed remains a key step that the designer must complete before his job is over. Sensitivity analysis evaluates the likely changes in the device's performance, usually due to element value tolerances or due to value changes with time and temperature.

Sensitivity analysis also determines the changes to be expected in the design's performance due to factor variations of uncontrollable character. If the design is found to be too sensitive, the designer

projects the *worst-case* scenario—to help plan for the unexpected (Pinel and Singhal, 1980). However, studies indicate that worst-case projections or conservative designs are often unnecessary and that a "robust design" can greatly reduce off-target performance caused by poorly controlled manufacturing conditions, temperature or humidity shifts, wider component tolerances used during fabrication, and also field abuse that might occur due to voltage/frequency fluctuations, vibration, etc. (Bagchi and Kumar, 1993).

Robust design should not be confused with rugged or conservative design, which adds to unit cost by using heavier insulation or high reliability, high tolerance components. As an engineering methodology robust design seeks to reduce the sensitivity of the product/process performance to the uncontrolled factors through a careful selection of the values of the design parameters. One straightforward way to produce robust designs is to apply the "Taguchi method" (Phadke, 1989; Bagchi, 1993).

The Taguchi method may be illustrated as follows. Suppose that a European product (chocolate bars) is to be introduced in a tropical country where the ambient temperature rises to 45°C. If the European product formulation is directly adopted, the result may be molten bars on store shelves in Bombay and Singapore and gooey hands and dirty dresses, due to the high temperature sensitivity of the European formula (Curve 1, Figure 8.2).

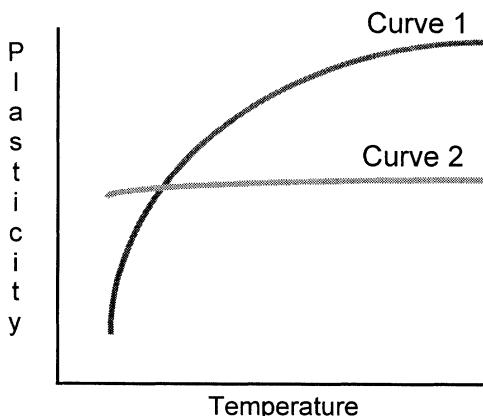


FIGURE 8.2 DEPENDENCE OF PLASTICITY ON AMBIENT TEMPERATURE

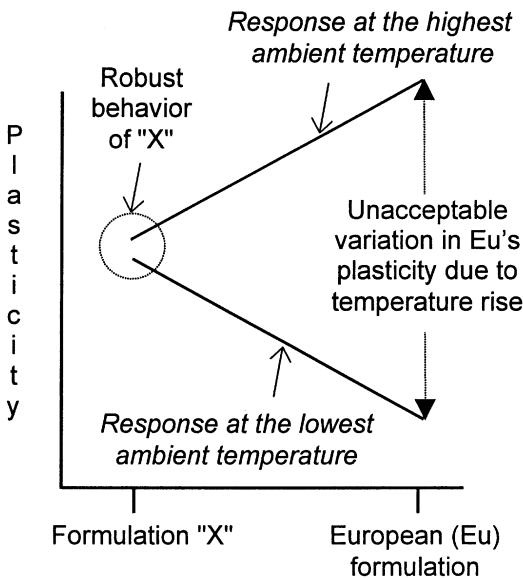


FIGURE 8.3 INTERACTION OF THE EFFECTS OF TEMPERATURE AND CHOCOLATE FORMULATION

The behavior of the bar's plasticity may be experimentally explored to determine its robustness to temperature, but few product designers actually attempt this. Taguchi would suggest that we do here some special "statistical experiments" in which both the bar's formulation (the original European and perhaps an alternate prototype formulation that we would call "X") *and* ambient temperature would be varied simultaneously and systematically and the consequent plasticities observed.

Taguchi was able to show that it is often possible to discover an alternate bar design (here an appropriate chocolate bar formulation) that would be robust to temperature. The trick, he said, is to uncover any "exploitable" *interaction* between the effect of changing the design (e.g. from the European formulation to Formulation "X") and temperature. In the language of statistics, two factors are said to interact when the influence of one on a response is found to depend on the setting of the other factor (Montgomery, 1996). Figure 8.3 shows such an interaction, experimentally uncovered. Thus, a "robust" chocolate bar may be created for the tropical market if the original European formation is changed to Formulation "X."

8.8.2 Robust Design by the "Two-step" Taguchi Method

Note that a product's performance is "fixed" primarily by its design, i.e., by the settings selected for its various design factors. Performance may also be affected by *noise*—environmental factors, unit to unit variation in material, workmanship, methods, etc., or due to aging/deterioration (Figure 8.4). The breakthrough in product design that Taguchi achieved renders performance robust *even in the presence of noise*, without actually controlling the noise factors themselves.

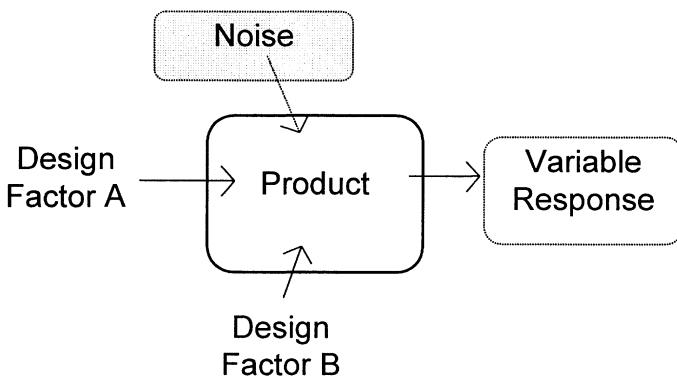


FIGURE 8.4 DESIGN AND NOISE FACTORS BOTH IMPACT RESPONSE

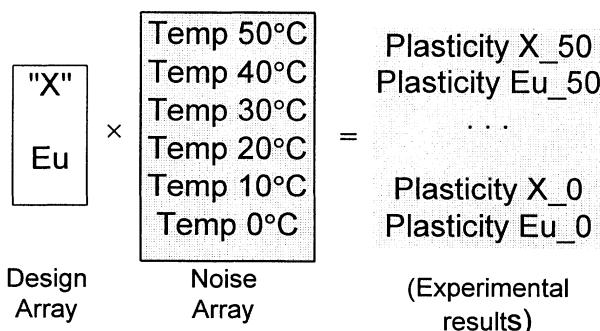


FIGURE 8.5 DESIGN×NOISE ARRAY EXPERIMENTS AND THEIR RESULTS

Taguchi's special "design \times noise array" experiments (Figure 8.5) discover those optimum settings. Briefly, the procedure first builds a special assortment of prototype designs (as guided by the "design array") and then tests these prototypes for their robustness in "noisy" conditions. For this, each prototype is "shaken" by deliberately subjecting it to different levels of noise (selected from the "noise array," which simulates noise variation in field conditions). Thus performance is studied systematically under noise in order to find eventually a design that is insensitive to the influence of noise.

To guide the discovery the "optimum" design factor settings Taguchi suggested a two-step procedure. In Step 1, optimum settings for certain design factors (called "robustness seeking factors") are sought so as to ensure that the response (for the bar, plasticity) becomes *robust* (i.e., the bar does not collapse into a blob at least up to 50°C temperature). In Step 2, the optimum setting of some *other* design factor (called the "adjustment" factor) is sought to put the design's average response at the *desired target* (e.g., for plasticity a level that is easily chewable).

For the chocolate bar design problem, the alternative design "factors" are two candidate formulations—one the original European, and the other that we called "X." Thus, the design array would contain two alternatives "X" and "Eu." "Noise" here is ambient temperature, to be experimentally varied over the range 0°C to 50°C, as seen in the tropics.

Figure 8.5 shows the experimental outcome of hypothetical design \times noise array experiments. For instance, response value Plasticity X_50 was observed when formulation X was tested at 50°C. Figure 8.3 is a compact illustrative display of these experimental results. It is evident from Figure 8.3 that Formulation X's behavior is quite robust even at tropical temperatures. Therefore, the adaptation of Formulation "X" would make the chocolate bar "robust," i.e., its plasticity would not vary much even if ambient temperature had wide swings.

Note, however, that Taguchi's "two-step" procedure works well in about 50% of actual design situations when design factor effects are *separable*, enabling identification of the robustness-seeking and the adjustment factors. Numerous other design projects such as unbreakable plastic parts (that must retain their toughness), robust

electronic devices (that must consistently deliver amplification, display of signals, error-free data transmission features, etc. as designed), or metallurgical processes (that must withstand wide variations in ore quality) must use more powerful modeling and optimization methods. (Note that the fitness landscape here may have many local optima.) We describe such other methods in the following sections.

8.8.3 Advanced QE Methods: Robust Design of an Electronic Device

A passive electronic filter device (Figure 8.6) is to be fabricated to enable measurement of displacement signals generated by a strain-gage transducer (Filippone, 1989; Suh, 1990). The filter will be mass-produced using inexpensive industrial quality components. The filter has two responses— ω_c , the cutoff frequency (target value 6.84 Hz) and a full-scale galvanometer deflection D (3.00 inches).

The designer here must specify the optimum values for the components R_2 , R_3 and C . The filter incorporates four other accessories V_s , R_s , R_g and G_{sen} (the sensitivity of the galvanometer), already existing. The filter interfaces the strain gage transducer-demodulator and a galvanometer-light beam deflection indicator.

We now turn to the designer's challenge: The filter is intended to be fabricated using inexpensive industrial quality components.

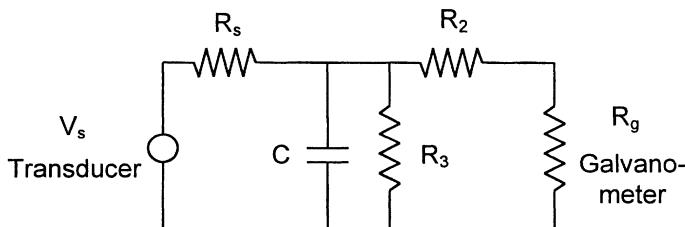


FIGURE 8.6 ELECTRONIC FILTER INTERFACING A STRAIN GAGE TRANSDUCER WITH A GALVANOMETER

Industrial quality electronic components—resistors, capacitors, etc.—are typically mass-produced and therefore inherit coarse tolerance. The coarse tolerance of components is actually an exasperating source of "noise" (high variability in unit-to-unit performance from nominal design objectives) for devices fabricated with such components. Electronic components may also vary from their nominal values due to the effect of uncontrolled environmental factors such as temperature etc., again contributing to "noise."

For the passive filter the application of Kirchoff's laws allows us to derive the mathematical expressions ((8.1 and 8.2)) relating the two performance characteristics of interest and design parameters as follows.

$$\omega_c = \frac{(R_2 + R_g)(R_s + R_3) + R_3 \cdot R_s}{2\pi(R_2 + R_g) \cdot R_3 \cdot R_s \cdot C} \quad (8.1)$$

$$D = \frac{|V_s| R_g R_s}{G_{sen}[(R_2 + R_g)(R_s + R_3) + R_s R_3]} \quad (8.2)$$

(8.1) and (8.2) actually *constrain* the total design space consisting of all possible values of R_2 , R_3 and C to the feasible set of solutions for which on-target performance ($\omega_c = 6.84$ Hz and $D = 3$ inches) would be *fait accompli*. Consequently, given the target values of ω_c (= 6.84 Hz), D (= 3.00 in) and a value of R_3 , the two remaining design parameters (R_2 and C) may be obtained as

$$R_2 = \frac{|V_s| R_g R_s - D G_{sen} R_3 + R_s}{D G_{sen} (R_3 + R_s)} - R_g \quad (8.3)$$

and

$$C = \frac{|V_s| R_g (R_3 + R_s)}{2\pi\omega_c R_3 (|V_s| R_g R_s - D G_{sen} R_3 R_s)} \quad (8.4)$$

The combination of R_3 , R_2 and C thus obtained (with $\omega_c = 6.84$ Hz and $D = 3$ inches) is a feasible design (though not yet robust or optimum), because it satisfies the target requirements for both ω_c and D . The next step is to search for a feasible design (here a value of R_3) that maximizes robustness. Equations (8.3) and (8.4) together define the contour in the design space on which one would conduct this search.

Consider first the task of making the cutoff frequency (ω_c) robust, i.e., ω_c of the fabricated filters must exhibit only a *minimum variability* from the target value (6.84 Hz). Thus, the designer must determine here the optimal R_2 , R_3 and C values that can make ω_c robust while also satisfying the two design requirements ($\omega_c = 6.84$ Hz and $D = 3.00"$).

The response functions for ω_c and D are obtainable from electronic circuit theory (equations (8.1) and (8.2)). You may observe that both (8.1) and (8.2) involve considerable degree of interaction among the effects of R_2 , R_3 and C . In other words, neither (8.1) nor (8.2) can be written as $f(R_2) + g(R_3) + h(C)$. In such situations we say that the effects of R_2 , R_3 and C on ω_c are not *additive*, or that they are not *separable*. So is the situation with D . A reference to the Taguchi literature tells us that the simple two-step design method will not work here (Phadke, 1989; Bagchi and Kumar, 1993). However, it is possible to separately make the response ω_c by a procedure called constrained optimization (Rao, 1991). Similarly, D can also be made robust separately. But this would not give Pareto optimality. We describe the procedure of constrained optimization in the following paragraphs.

The "design array" in this problem is quite simple: Since (8.3) and (8.4) together restrict the values that the three design parameters R_2 , R_3 and C can take together (to ensure the production of a design that satisfies the target constraints (8.1) and (8.2), or in other words, is feasible), R_3 is the only design parameter to select. The design array is therefore all realistic values of R_3 . The "noise array" is an L_8 orthogonal array (Montgomery, 1997) that would simulate the "noisy" assembly conditions arising due to the fluctuating industrial quality component tolerances. These tolerances are shown in Table 8.1. Table 8.2 shows the L_8 array—the noise array—for this problem. To maximize the robustness of ω_c and D , we will consider the minimization of two separate objective functions. The first one is the *noise-induced variance* of cut-off frequency $Var(\omega_c)$ given by (8.7) while

the second objective is the *noise-induced variance* of deflection $Var(D)$ given by (8.8), each of which is evaluated by using the noise array.

Observe that if you specify R_3 , this virtually completes that filter design task because the other two design parameters R_2 and C may be found using (8.3) and (8.4) while components R_s , R_g , G_{sen} and V_s remain unchanged at their prescribed values. To evaluate such a design's robustness, what remains to be done is that this design be "shaken" by tolerance-induced noise and its effect on the two responses observed. Such "shaking" can be simulated in an orderly way, as Phadke (1989) has shown, when guided by the noise array (see Figure 8.5).

The noise array for the present design problem shown in Table 8.2 is a means to simulate tolerance-perturbed values of R_2 , R_3 , C , R_s , R_g , G_{sen} and V_s , all the components in the electronic filter device (Figure 8.6). The "-" value under R_3 in the first row (Experiment #1, Table 8.1), for instance, indicates that R_3 would have a value in this experiment that is 5% less than the nominal value of R_3 specified by the designer. Similarly, R_2 will have a value in Experiment #1 that is 5% less than the value of C obtained from (8.4). Similarly, the appropriate values of C , R_s , R_g , etc. are obtained from Table 8.1. The two responses (ω_c and D) are then "observed" for Experiment #1 by substituting these values for R_2 , R_3 , C , R_s , R_g , G_{sen} and V_s in (8.1) and (8.2) to produce $\omega_{c(obs),1}$ and $D_{(obs),1}$.

A similar procedure using the settings in Row 2 of Table 8.2 (Experiment #2) would produce $\omega_{c(obs),2}$ and $D_{(obs),2}$, and so on, till all eight perturbed values for ω_c and D have been found. The *sample means* for the eight experiments and the simulated variance for the two responses are calculated as follows.

$$Avg - \omega = \sum_{i=1}^8 \omega_{c(obs),i}$$

$$Var(\omega_c) = \frac{1}{(8-1)} \sum_{i=1}^8 [\omega_{c(obs),i} - \omega_{c,mean}]^2$$

$$Avg_D = \sum_{i=1}^8 D_{(obs),i}$$

$$Var(D) = \frac{1}{(8-1)} \sum_{i=1}^8 [D_{(obs),i} - D_{mean}]^2$$

The above process thus simulates the fabrication of the electronic filter using some special combinations of coarse tolerance components. Phadke (1989) has discussed how orthogonal arrays (such as the eight rows in Table 8.2) are enough to produce a representative sample of "noisy" responses in robust design studies.

TOLERANCE 8.1: TOLERANCE LEVELS THAT CAUSE NOISE

Components	"-"	Nominal Level	"+"
R ₃ (Ω)	R ₃ - 5%	R ₃	R ₃ + 5%
R ₂ (Ω)	R ₂ - 5%	R ₂	R ₂ + 5%
C (μF)	C - 5%	C	C + 5%
R _s (Ω)	119.82	120	120.18
R _g (Ω)	97.853	98	98.147
G _{sen} (μV/in)	656.594	657.58	658.566
V _s (V)	0.014978	0.015	0.015023

TABLE 8.2: L8 ORTHOGONAL ARRAY TO SIMULATE NOISE

Expt #	R ₃	R ₂	C	R _s	R _g	G _{sen}	V _s	ω _{c(obs)}	D _(obs)
1	-	-	-	-	-	-	-		
2	-	-	-	+	+	+	+		
3	-	+	+	-	-	+	+		
4	-	+	+	+	+	-	-		
5	+	-	+	-	+	-	+		
6	+	-	+	+	-	+	-		
7	+	+	-	-	+	+	-		
8	+	+	-	+	-	-	+		

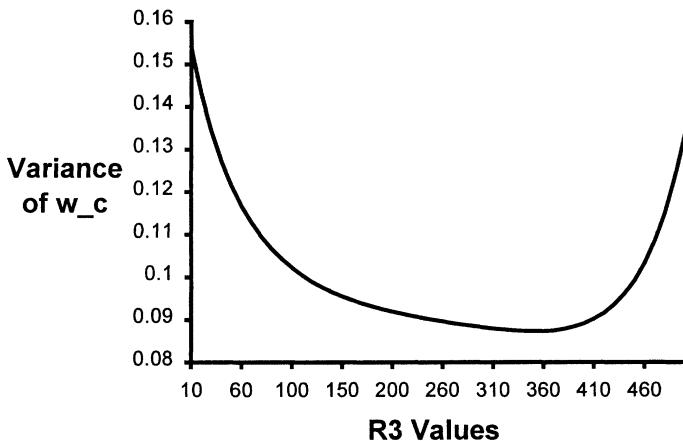


FIGURE 8.7 VARIANCE OF ω_c ESTIMATED BY DESIGN×NOISE ARRAY EXPERIMENTS

Figure 8.7 displays the results of constrained optimization. The (lack of) robustness of ω_c is the noise-induced variance of ω_c , calculated by conducting simulated design×noise array experiments. The Y-axis shows $\text{variance}(\omega_c)$ produced by fluctuations in component values at various settings of the design parameter R_3 . Figure 8.7 shows that robustness of ω_c is maximum (this is where the variability in ω_c caused by uncontrolled component tolerances is minimum) when R_3 is near 350 ohms. Such methods raise the "quality" of manufactured products without using tight-tolerance, high-precision parts.

Without the use of the above procedure, the cutoff frequency of the filter would remain widely variable when low cost industrial components are used to fabricate the filter. Figure 8.8 shows the results of a similar procedure when the robustness of D (the full-scale galvanometer deflection) rather than that of ω_c is sought. Here the optimum value of R_3 is about 150 ohms.

Note, however, that the filter design problem could not be tackled using Taguchi's "two-step" procedure (as done for the chocolate bar design) because the effects of R_2 , R_3 and C on ω_c (or even D) are not separable. As mentioned earlier, *separability* of factor effects is a must for Taguchi's simple two-step robustness-seeking method to work (Taguchi, 1986; Phadke, 1989).

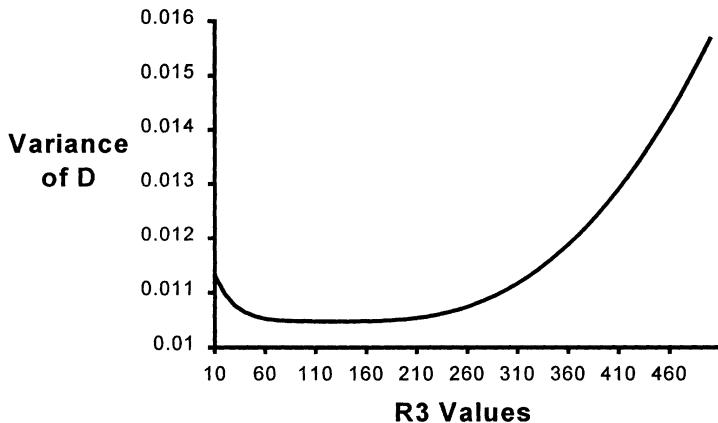


FIGURE 8.8 VARIANCE OF D ESTIMATED BY DESIGN×NOISE ARRAY EXPERIMENTS

Still, the above solutions are two *single-objective* robust designs. We have not found the Pareto optimal solutions yet. The final choice of R_3 would therefore depend on resolving the multiple- (two-) objective problem: (1) maximize the robustness of cut-off frequency ω_c and also (2) maximize the robustness of deflection D.

As we saw above, constrained optimization can find the two single-objective "best" designs in this problem. Using equations (8.1) and (8.2) it is possible to first maximize the robustness of ω_c , and then separately maximize the robustness of D. A procedure for dual response systems based on the response surface methodology (RSM) is such a methodology (Draper and Smith, 1998). A fundamental limitation of RSM-type methods, however, is that these use locally fitted response surfaces, usually quadratic, and repeated updates of these surfaces as an optimal value is approached. This works very well with smooth, well-behaved response surfaces or functions (examples are Figures 8.7 and 8.8), but less well or not at all with irregular surfaces involving the presence of local optima. As opposed to this, heuristic search methods such as the SGA can easily handle such irregularities. To apply SGA, R_3 would be generated as the decision variable (a chromosome). R_2 , C and the other components would be found as described above. The two objective functions $Var(\omega_c)$ and $Var(D)$ would be found using Table 8.2 and the expressions given on page 207. Thus, here SGA would evolve

different values of R_3 . Shaking by noise, however, would continue to be provided by the "noise array" as before.

Using SGA and working with one objective at a time we produced the two extreme solutions shown below. (Try to compare these results to Figures 8.7 and 8.8.)

The design maximizing the robustness of ω_c

$R_3 \Omega$	$R_2 \Omega$	$C \mu\text{f}$	Avg ω_c Hz	Avg D in.	Var ω_c	Var D
341	7.69	480	6.841	3.004	.0873	.012

The design maximizing the robustness of D

$R_3 \Omega$	$R_2 \Omega$	$C \mu\text{f}$	Avg ω_c Hz	Avg D in.	Var ω_c	Var D
140	181	437	6.844	3.003	.0996	.0104

The two above solutions confirm that the two objectives $Var(\omega_c)$ and $Var(D)$ that we are trying to minimize do not have a common solution. In fact, these two objectives are in conflict, hence a Pareto-type solution must be sought here.

Before we end this section, we must highlight a critical limitation of the constrained optimization procedure. We produced above the two "best" robust designs but none is "globally optimum." This is due to the manner in which constrained optimization works. Constrained optimization can work with only *one* objective at one time, or at best with a "weighted average" or weighted sum of several objectives. It has no way to directly optimize *two or more independent objectives simultaneously*. In the following pages we counter this difficulty.

8.8.4 Application of NSGA to the Filter Design Problem

Sharma (1996) originally applied NSGA to the above electronic filter design problem. In Sharma's method, the two conflicting objectives (8.7) and (8.8) were simultaneously minimized in the Pareto optimal sense, by varying the single decision variable R_3 as guided by NSGA. The NSGA parameters used by Sharma were population size (p_s) = 100, crossover probability (p_c) = 0.75 and mutation probability (p_m) = 0.05—values determined using a pilot *design of experiments* (DOE)

study to maximize the rate of the NSGA's convergence (Bagchi and Deb, 1996). The other parameters were string length = 32, R_3 (minimum) = 1 Ω , R_3 (maximum) = 350 Ω and σ_{share} for niche formation kept at 0.1. A small value of σ_{share} used ensured that even closely located Pareto optimal solutions would show up in the final solution set. Sharma did not use speciation, i.e., he did not impose any mating restriction.

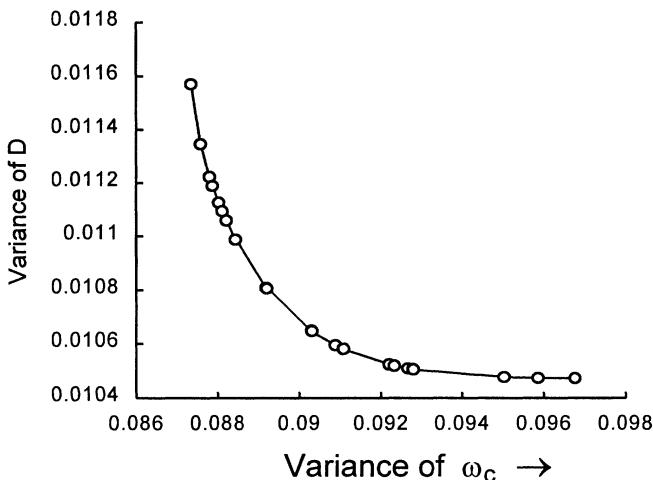


FIGURE 8.10 TWENTY FIVE "FIRST RANK" ROBUST DESIGNS ON THE PARETO-OPTIMAL FRONT

Twenty-five "first rank" solutions (the members of the Pareto-optimal front) selected at random from Generation 50 are shown in Figure 8.10 with typical solutions shown in Table 8.3. It is easy to verify that if any one of these solutions is better than (i.e., dominates) any other solution with respect to one objective (say $Var(\omega_c)$), then it is worse with respect to the other objective ($Var(D)$) and vice-versa. This is the property of *non-dominance* and it affirms Pareto-optimality of these solutions.

Note that no method presently exists to identify Pareto optimal robust designs.

**TABLE 8.3 SAMPLE PARETO-OPTIMAL ROBUST DESIGNS
PRODUCED BY NONDOMINATED SORTING GA**

R3 (Ω)	R2 (Ω)	C (farad)	Avg(ω_c) (Hz)	Var(ω_c)	Avg(D) (in)	Var(D)
311.7881	22.43919	0.000462	6.841476	0.087877	3.00349	0.011188
156.2765	157.7785	0.000434	6.844165	0.095864	3.003232	0.010474
306.0652	25.66875	0.000458	6.841552	0.088029	3.003469	0.011125
195.026	111.5562	0.000424	6.8434	0.092806	3.00325	0.010506
225.7759	82.24873	0.000426	6.842843	0.091095	3.00328	0.010581
195.1129	111.4653	0.000424	6.843399	0.092801	3.00325	0.010507
299.6366	29.40162	0.000454	6.841641	0.088213	3.003447	0.011058
269.8776	48.28555	0.000439	6.842093	0.089215	3.003361	0.010806
292.3232	33.78978	0.000450	6.841746	0.088438	3.003423	0.010988
341.029	7.18926	0.000483	6.841166	0.087348	3.003618	0.011571
197.1142	109.3861	0.000424	6.843362	0.092675	3.003252	0.010510
225.9195	82.1241	0.000426	6.842840	0.091088	3.003279	0.010581
147.579	169.993	0.000438	6.844350	0.096766	3.003229	0.010473
324.7787	15.4167	0.000471	6.841321	0.087583	3.003543	0.011346
314.7266	20.81395	0.000464	6.841440	0.087804	3.003502	0.011222
337.4986	8.92698	0.000480	6.841196	0.087384	3.003601	0.011520
204.7699	101.6687	0.000424	6.843220	0.092214	3.003258	0.010524
229.909	78.70221	0.000427	6.842768	0.090896	3.003285	0.010595
303.139	27.35382	0.000456	6.841593	0.088111	3.003458	0.011094
242.519	68.38161	0.000430	6.842550	0.090320	3.003304	0.010647
340.9459	7.22985	0.000483	6.841167	0.087349	3.003618	0.011570
270.2351	48.0416	0.000439	6.842086	0.089201	3.003361	0.010808
242.6987	68.23972	0.000430	6.842547	0.090312	3.003305	0.010648
165.1582	146.0743	0.000430	6.843982	0.095038	3.003235	0.010478
202.6415	103.7775	0.000424	6.843259	0.092339	3.003255	0.010520

8.9 CHAPTER SUMMARY

In this chapter we described a special GA called the nondominated sorting genetic algorithm or NSGA. NSGA operates similar to SGA, except the way in which selection is done. NSGA uses a surrogate fitness function constructed after the population has been ranked in the order of nondominance.

NSGA tackles multiobjective optimization problems and produces a population of Pareto optimal solutions. To do this NSGA uses two important processes observed in natural evolution, namely, niche formation and speciation. Niche formation is how organisms share limited resources and avoid head-to-head competition to ensure their own survival. Speciation is the process by which distinct species are formed by isolation or by differential reproduction.

A bi-criteria problem—a robust engineering design problem—was solved in this chapter to illustrate the use of NSGA. In the next chapter NSGA is used to sequence jobs in a multiobjective flowshop.

9

MULTIOBJECTIVE FLOWSHOP SCHEDULING

It was mentioned in Chapter 4 that in sequencing jobs in a flowshop we are likely to confront several different (and often *conflicting*) management objectives. Consequently, a schedule may have to be evaluated by different types of performance measures. Some of these measures may give importance to completion time (e.g. makespan), some to due date (e.g. mean tardiness, maximum tardiness), and some others to the speed with which the jobs flow (e.g. mean flow time). The simultaneous consideration of these objectives is a multiobjective optimization problem. But, even for a single objective, flowshop sequencing is *NP-hard*. Therefore, solving even a *single* objective flowshop problem involving only 15-20 machines and 30-40 jobs by classical optimization methods would be quite difficult.

This chapter gives a brief description of the recently proposed approaches to solve the multiobjective flowshop problem. It then illustrates the steps for using GAs to sequence the multiobjective flowshop using a tri-objective flowshop problem.

9.1 TRADITIONAL METHODS TO SEQUENCE JOBS IN THE MULTIOBJECTIVE FLOWSHOP

A common difficulty with multiobjective optimization is the appearance of "objective conflict" (Hans, 1988): none of the feasible solutions achieves simultaneous optimization of makespan, mean flow time, mean tardiness, machine utilization, etc. in a flowshop, for example. In other words, the individual optimal solutions of each objective are usually different. Thus, a "most favored" solution would

be one that would cause *minimum* objective conflict. Such solutions may be viewed as points in the solution space that satisfy the priorities associated with the different objectives. Any of the solution methods described in Section 6.4 may be applied here.

To find such points, most classical methods would scalarize the objective vector (which is made up of several objectives) into one single objective. If necessary, SGA (the simple genetic algorithm) can be used at this point. Three such common methods that have been applied to the flowshop—the method of objective weighting, the method of distance functions, and the method of min-max formulation—are recalled below.

9.1.1 Method of Objective Weighting

This is probably the simplest of all classical multiobjective methods. In it multiple objectives such as makespan minimization, mean flow time minimization and mean tardiness minimization are combined with the help of subjectively fixed weight factors into a single objective function, Z , as follows:

$$Z = \sum_{i=1}^N w_i f_i$$

The weight factors $\{w_i\}$ in Z are fractional numbers ($0 \leq w_i \leq 1$) and all weights together sum up to 1.0. Clearly, the weight vector $(w_1, w_2, w_3, \dots, w_N)$ controls the optimal solution. This poses a decision problem in itself. Mathematically, a solution obtained with equal weights to all objectives may offer least objective conflict, but since the real world often demands a more "satisfactory" solution, priority information must be induced from the decision maker in the formation of the objective Z .

Objective weighting continues to be popular in management science (see Shang and Tadikamalla, 1998). The advantage of using the objective weighting technique is that the analyst can control the relative emphasis of one objective weight over another. Nevertheless, note that objective weighting does not guarantee Pareto optimality.

9.1.2 Method of Distance Functions

This method has also been applied to solve the multiobjective flowshop scheduling problem. In this method, scalarization (conversion of multiple objectives into a single objective) is achieved by using a *demand-level vector* (\bar{y}), which is defined as the (vector) statement of *goals* being sought for each distinct objective. The decision maker must specify the demand level vector *a priori*. The single objective function thus formed from the individual objectives takes the following form:

$$Z = \left[\sum_{i=1}^N |f_i - \bar{y}_i|^r \right]^{1/r}, \quad 1 \leq r \leq \infty,$$

Usually an Euclidean metric ($r = 2$) is chosen, with \bar{y} being made up of individual optima of the different objectives (Hans, 1988). The solution obtained by solving this restated problem is obviously dependent on the specified value of \bar{y} . Arbitrary selection of a demand level is considered to be highly undesirable; a wrong demand level will often lead to a non-Pareto optimal or a dominated solution, if at all.

The distance function method is similar to the method of objective weighting. The only difference is that in this method the goal (the individual optima) for each objective function must be specified whereas in the objective weighting method the relative importance of each objective needs to be specified, before optimization is attempted. Note further that the distance function method does not guarantee Pareto optimality even if Z is successfully minimized.

9.1.3 The Min-Max Formulation

This method has also been applied to sequence the multiobjective flowshop. It is a variation of the distance function method in the sense that it attempts to minimize the *relative deviations* of each single objective function from its own individual optimum. That is, it tries to minimize objective conflict. For a minimization problem, the corresponding min-max problem is formulated as follows:

Minimize $F(x) = \max [Z_j(x)], j = 1, 2, \dots, N$

$Z_j(x)$ is calculated for the individual nonnegative target optimal values ($\bar{f}_j > 0$) as follows:

$$Z_j(x) = \frac{f_j(x) - \bar{f}_j}{\bar{f}_j}, \quad j = 1, 2, \dots, N$$

This method yields the best possible "compromise solution" when objectives $\{f_j(x)\}$ with *equal* priority are required to be optimized.

9.2 DISADVANTAGES OF CLASSICAL METHODS

In each of the above methods, multiple objectives were combined to form one objective by using some knowledge of the problem being solved. The optimization of this single objective may approach a near-Pareto optimal solution (a desirable characteristic of a multiobjective solution, see Goldberg (1989), page 201), but it results in a *single point* solution. In practice, the shop manager often seeks to meet a variety of goals and hence he/she needs some means to *compare* several different alternatives, all of which seem acceptable otherwise.

Besides, some classical methods are also effort-intensive for the analyst for they require the knowledge of the individual optima prior to one's attempting (vector) optimization. But perhaps the most serious drawback of the classical methods is their sensitivity to weight choices or demand (goal) levels. The manager looking for a multiobjective solution based on these methods therefore must provide an explicit statement of the priority of each objective *a priori* to enable the analyst to form the single (scalar) objective from the set of objectives at hand. As noted, this is a critical step because the solutions obtained may be very dependent on the underlying weight vector or demand vector. Consequently, if classical methods are used, then for different priority situations or for different weight vectors the same problem must be solved several times. (See Fonseca and Fleming, 1995.)

9.3 ADAPTIVE RANDOM SEARCH OPTIMIZATION

The adaptive random search optimization technique (ARSO, Beale, 1977 and Beale and Cook, 1978) is a multiobjective optimization method that needs a starting point in the solution space being searched. The method then proceeds to try to improve upon this initial solution by perturbing the decision parameters. Statistics (mean and variances) are maintained for all perturbations that produce improvements. An ARSO solution is said to improve upon another solution if it dominates the old one. The statistics are used to guide the direction of future perturbations.

Such randomized perturbation techniques have been shown to solve a large class of optimization problem *faster* than gradient techniques when the number of parameters (decision variables) exceeds 4. The convergence time of such methods increases linearly with the number of decision variables being manipulated.

Attempts to develop Pareto optimal solutions to multiobjective flowshops have been relatively few. As stated earlier, such solutions are the problem's nondominated or "efficient" solutions and any other solution need not be considered by the decision maker. It should be clear that it is much easier for the decision maker to apply his/her subjective preferences to select the "best" solution from these "efficient" solutions. This is clearly better than discovering later that a selected solution is dominated by another solution not originally in the decision maker's list.

In the following sections we review the use of NSGA to determine Pareto optimal solutions to the multiobjective flowshop problem.

9.4 RECOLLECTION OF THE CONCEPT OF PARETO-OPTIMALITY

The general multiobjective optimization problem contains a number of objectives while the solution(s) must also satisfy a number of inequality and equality constraints. Mathematically, the problem may be stated as follows.

Minimize/Maximize $f_i(x) \quad i = 1, 2, \dots, N$

Subject to $g_j(x) \leq 0 \quad j = 1, 2, \dots, J$

$h_k(x) = 0 \quad k = 1, 2, \dots, K$

Here the decision parameter x is a p -dimensional vector made up of p decision variables. Solutions to a multiobjective optimization problem are mathematically expressed in terms of nondominated or "superior" points.

The nondomination property of solutions may be explained as follows. In a minimization problem, a solution vector $x^{(1)}$ is partially dominated by another vector $x^{(2)}$ (written as $x^{(1)} \prec x^{(2)}$), when no component value of $x^{(2)}$ is less than $x^{(1)}$ and at least one component of $x^{(2)}$ is strictly greater than $x^{(1)}$. In a minimization problem if $x^{(1)}$ is partially less than $x^{(2)}$, we say that the solution $x^{(1)}$ *dominates* $x^{(2)}$ or the solution $x^{(2)}$ is inferior to $x^{(1)}$ (Section 6.6). Any member of such vectors that is not dominated by any other member is said to be *nondominated* or *noninferior* (Figure 9.1).

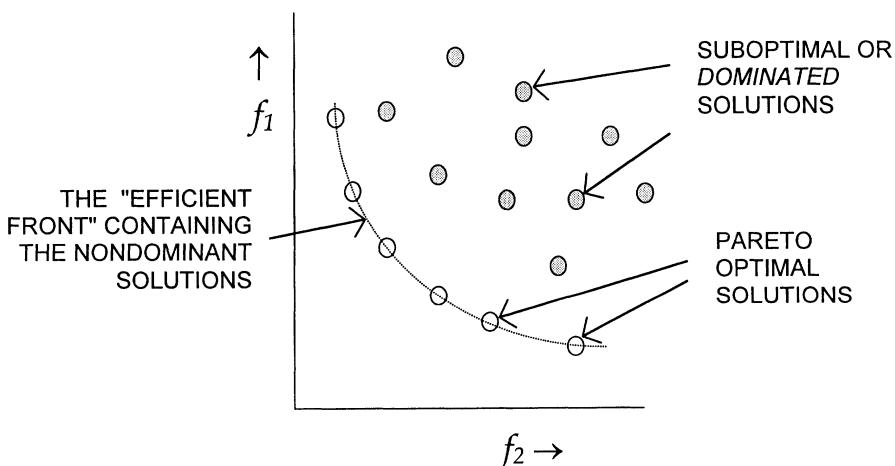


FIGURE 9.1 THE EFFICIENT FRONT FOR A BI-OBJECTIVE MINIMIZATION PROBLEM

Although the concept of Pareto optimality is now over ninety years old (Pareto, 1906), methods for finding nondominated solutions are still relatively few. In the following section we harness NSGA, the multiobjective GA described in the previous chapter, to find efficient solutions to the multiobjective flowshop. We solve below a collection of sample problems of differing sizes.

9.5 NSGA SOLUTIONS TO THE MULTIOBJECTIVE FLOWSHOP

The kernel of NSGA is the *ranking selection* method it uses to emphasize the Pareto optimal regions where the desirable solutions reside. NSGA also uses a niche forming procedure to maintain a stable population of good solutions (thus avoiding genetic drift). Thus NSGA differs from SGA in the manner the *selection* operator works. The crossover and mutation operators in NSGA work as they do in SGA, so the representation (solution coding) schemes can be identical in SGA and NSGA. However, before selection is performed in NSGA, the population is ranked on the basis of the nondominated sorting concept (see the flow chart in Figure 8.1), to emphasize Pareto-optimality.

Suppose we wish to minimize makespan, mean flow time as well as the mean tardiness of jobs in a flowshop *simultaneously*. The solutions (permutation representation of job sequences) may then be coded, for instance, by the procedure we described in Section 4.6.1. Crossover would be "one point" while mutation would be adjacent job-interchange. Reproduction would give preference to nondominated members (identified by nondomination ranking of all solutions in the population). Sharing would be phenotypic, derating the dummy fitness values of the solutions by dividing it by niche count (Section 8.7.3).

NSGA would be initiated with a randomly picked collection (population) of size p_s comprising different permutations involving n jobs. Table 4.1 displays the processing times and the due dates for a plant scale flowshop problem in which 49 jobs are to be optimally sequenced in a 15-machine flowshop. $49!$ different solutions are possible here from which the nondominating solutions must be separated. The three objectives to be simultaneously minimized are

makespan, mean flow time and mean tardiness of jobs. In this illustration, the NSGA was optimally parameterized using a design-of-experiments approach (Chapter 3) using pilot runs. This determined the optimal parameter values to be population size (p_s) = 200, probability of mutation (p_m) = 0.0, probability of crossover (p_c) = 0.9 and $\sigma_{\text{share}} = 1$.

**TABLE 9.1 NUMBER OF PARETO OPTIMAL SOLUTIONS
FOUND BY NSGA FOR THE 15 m/c-49 Job
FLOWSHOP PROBLEM OF TABLE 4.1**

Generation →	0	25	50	75	100	125	150	175	200	225	250
Seed # 1	7	7	16	15	25	36	29	26	27	27	27
Seed # 2	2	8	16	10	15	26	25	23	26	23	23
Seed # 3	3	11	3	18	20	21	11	16	12	11	12
Seed # 4	2	15	12	27	18	36	34	35	30	29	26
Seed # 5	7	7	15	21	13	24	25	26	26	26	26
Seed # 6	8	12	14	5	17	22	24	24	27	26	26
Seed # 7	11	9	10	14	38	30	33	31	31	33	32
Seed # 8	7	7	4	8	15	14	16	20	14	15	16
Seed # 9	12	9	10	21	20	29	30	31	29	27	27
Seed # 10	5	4	11	16	19	23	23	18	16	19	17
Seed # 11	5	18	15	19	17	7	5	6	5	6	6
Seed # 12	3	15	20	17	12	9	14	12	12	13	13
Seed # 13	8	11	12	19	11	22	30	30	30	30	30
Seed # 14	7	7	15	5	8	10	11	10	10	9	9
Seed # 15	6	8	9	25	25	23	29	34	43	45	44
Seed # 16	7	11	6	7	7	16	15	14	13	14	15
Seed # 17	5	2	10	14	22	31	41	43	41	36	38
Seed # 18	11	13	12	20	20	19	30	20	21	18	15
Seed # 19	7	18	16	14	25	49	52	55	58	60	58
Seed # 20	4	15	10	27	17	22	19	19	18	18	18
Seed # 21	7	19	9	24	30	38	41	40	32	37	36
Seed # 22	9	11	8	9	11	17	10	9	5	10	12
Seed # 23	9	9	20	13	23	28	31	29	31	29	28
Seed # 24	3	15	8	21	14	7	18	14	12	8	9
Seed # 25	10	14	11	12	11	21	23	23	22	22	21
Seed # 26	5	11	16	19	10	20	18	19	20	20	17
Seed # 27	6	11	8	15	14	15	21	20	20	21	22
Seed # 28	7	11	16	20	10	25	36	36	38	38	39
Seed # 29	4	4	9	4	10	22	13	11	18	18	17
Seed # 30	6	13	15	3	8	15	21	22	24	24	21
Seed # 31	3	10	14	8	6	4	4	4	4	4	4
Seed # 32	5	14	13	11	10	12	7	8	9	9	10
Seed # 33	4	12	11	18	37	55	53	44	50	45	49
Seed # 34	4	9	7	25	20	17	34	29	10	13	13
Seed # 35	9	9	5	10	18	23	8	8	7	7	7
Seed # 36	7	13	7	17	27	10	14	9	7	7	7
Seed # 37	4	12	20	17	32	32	28	26	27	30	33
Average Count	6.2	10.9	11.7	15.4	17.7	22.4	23.7	22.8	22.3	22.4	22.2

FIGURE 9.2 GROWTH IN PARETO OPTIMAL SOLUTIONS FOUND BY NSGA FOR DIFFERENT RANDOM STARTS

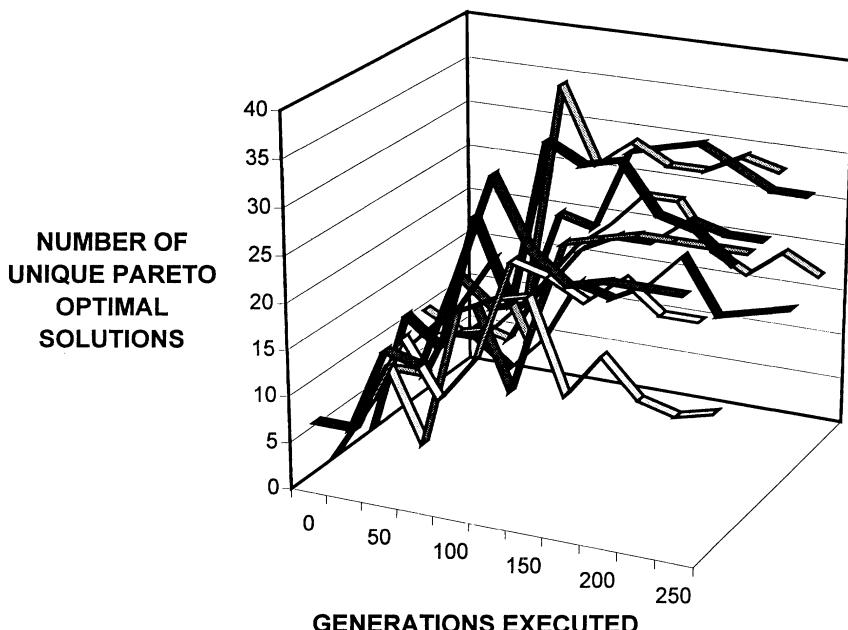


Table 9.1 presents the results of running NSGA for 250 generations with 37 different initial random populations. Figure 9.2 displays the growth in the number of Pareto optimal solutions found for a few randomly picked starting seeds in Table 9.1 as the function of generations evolved by NSGA.

9.6 HOW NSGA PRODUCED PARETO OPTIMAL SEQUENCES

As Table 9.1 indicates, nearly 22% of the population at the end of 250 generations were nondominated or Pareto optimal. We recount briefly how NSGA achieved this and then mention how the quality of the solutions may be further improved. The two key strategies employed by NSGA are (1) the use of nondominated (or Pareto) ranking to seek individuals with the best Pareto ranks, and (2) the use of fitness sharing to obtain as many nondominated solutions as possible.

For each NSGA iteration, an initial set of solutions (randomly generated by a seed) started the process. The nondominated individuals among these solutions were first identified from the initial population of solution. These initial nondominated individuals constitute the *first nondominated front* in the population and assigned a dummy fitness value proportional to the population size (dummy fitness = N). This dummy fitness is intended to give an equal reproductive chance to all these initial *nondominated* individuals.

Diversity among the solutions was maintained by *sharing* fitness value among the individuals (see Section 8.3). Next, the individuals in the first front were ignored temporarily and the rest of the population was attended to, as follows: Individuals on the *second* front were identified. These second front solutions were then assigned a new fitness value that was kept *smaller* than the shared dummy fitness value of the solutions on the first front. This was done to differentiate between the members of the first front and the members of the second front. Then, sharing was again done within the second front, and the process went on till whole population had been evaluated, and classified into successive fronts. This process led to the creation of several successive fronts of "nondominated" individuals.

Next, individuals in the whole population were reproduced according to their relative (dummy) fitness value. This approach facilitates the search for the nondominated regions of the Pareto-optimal fronts. This results in the quick convergence of the population towards the nondominated region while sharing helps to distribute the individuals over the entire nondominated region.

Figure 9.3 displays the impact of NSGA operating on the initial randomly generated solution population (marked in the figure by "0"). As the NSGA operated, *each* objective (makespan, mean flow time and tardiness) was minimized by Pareto domination sorting-based selection. After 250 generations, many Pareto solutions got separated. These appear (marked by "1") at the left bottom corner of the figure.

The efficiency of NSGA lies in the manner it reduces multiple objectives to be optimized to *a single dummy fitness function* using nondominated sorting procedure. If a solution is locally dominated, it is globally dominated. However the converse is not true. In each generation NSGA finds locally nondominated points only.

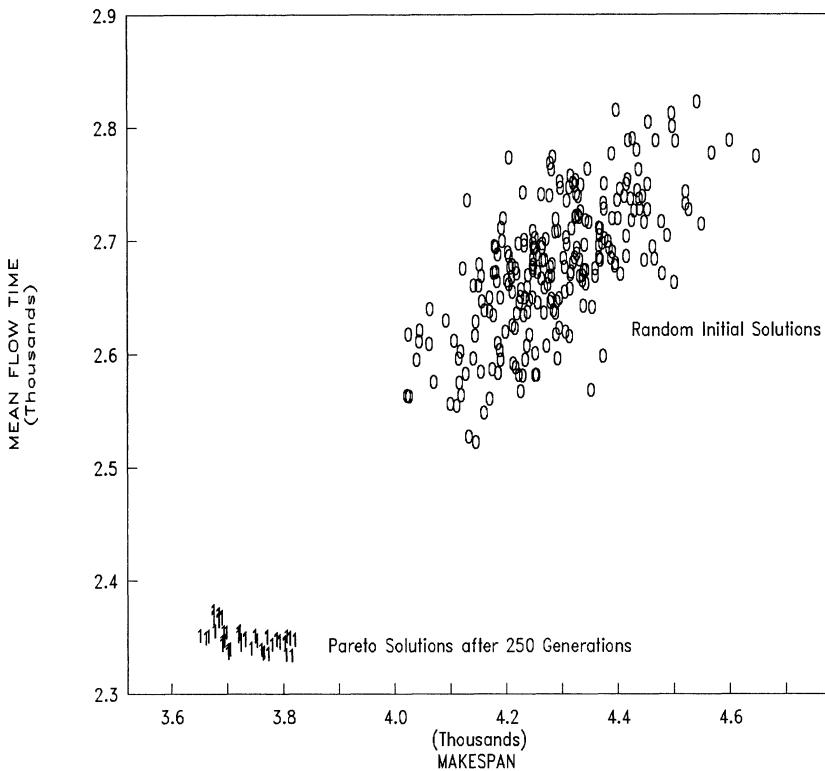


FIGURE 9.3 CONVERGENCE OF INITIAL RANDOM SOLUTIONS TO PARETO OPTIMAL TRI-OBJECTIVE SOLUTIONS IN 250 GENERATIONS OF NSGA EXECUTION

Still, if there exists any globally nondominated member, NSGA increases the likelihood of its survival. This is because in the NSGA scheme, any such individual will have the highest possible dummy fitness. Thus the power of NSGA lies in the successful transformation of a multiobjective problem, no matter how many objectives there are, to a single function problem, without losing the concept of vector optimization. By selecting nondominated points, NSGA actually processes the schemata that represent Pareto-optimal regions. Therefore, the building blocks for NSGA will be those schemata that represent the characteristics of globally nondominated individuals.

9.7 THE QUALITY OF THE FINAL SOLUTIONS

The actual quality of the solutions on the Pareto front may be further improved by merely increasing iterations, due to the effect of *fitness sharing*. The effect of sharing in GA is in some sense similar to that of mutation in that sharing promotes local neighbourhood search (Goldberg, 1989). The rate at which Pareto optimal solutions were discovered by NSGA for the 49-job 15-machine flowshop problem solved here are indicated by the rising ribbons in Figure 9.2.

The actual number of Pareto optimal solutions discovered by NSGA is a function of the Pareto optimal landscape (the number of solutions that exist for a problem), and population size (p_s). Experimental evidence indicates that the larger the number of jobs being sequenced, the higher should be the value of p_s to facilitate the finding of as many Pareto optimal solutions as possible for a given number of NSGA generations executed. We note here that the time to execute NSGA with a population size of 100 for 250 generations for the 49-job 15-machine problem on a HP 9000/850 system running compiled C++ was about 35 seconds.

NSGA implemented here did not use mating restriction, a phenomenon that encourages *speciation* (formation of species among solutions). Deb and Goldberg (1989) have shown that speciation improves NSGA's on-line performance (the rate at which GA converges to optimal solutions). This would be another way to improve the quality of the final solutions for a given number of generations executed. Deb and Goldberg recommend the use of phenotype sharing here, based on dummy fitness.

Yet another method based on elitism can improve the rate at which the Pareto optimal solutions are found. This method is described in the next chapter. In Chapter 11 this new method is statistically compared with NSGA using an assortment of tri-objective flowshop problems as testbeds.

9.8 CHAPTER SUMMARY

Classical multiobjective flowshop sequencing methods require *a priori* problem information for their solution. The method of distance functions (Section 9.1.2) requires the knowledge of the individual

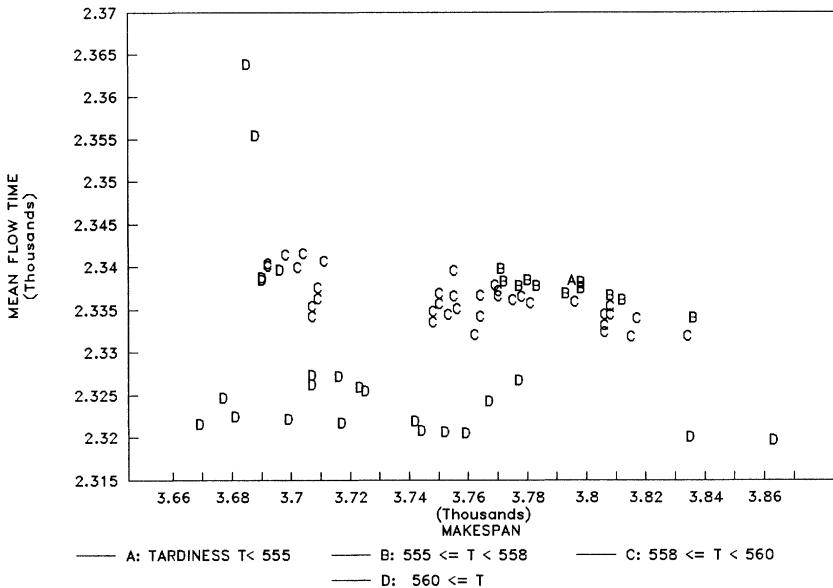


FIGURE 9.4 THE RELATIVE LOCATION OF THE PARETO OPTIMAL SOLUTIONS IN THE TRI-OBJECTIVE DECISION SPACE

optima. Yet, it does not guarantee reaching Pareto optimality. VEGA (Vector Evaluated Genetic Algorithm) suffers from the drawback that it cannot maintain stable and uniform reproductive potential across nondominated individuals. On the other hand, NSGA (Nondominated Sorting Genetic Algorithm), a methodology based on fitness sharing and nondominated sorting proposed by Srinivas and Deb (1995), can find a good number of Pareto optimal solutions quickly. Figure 9.4 displays the relative location of the Pareto optimal solutions found for the tri-objective 49-job 15 m/c flowshop example.

Some enhancements to NSGA can further improve its effectiveness. One such important enhancement is developed in the next chapter.

10

A NEW GENETIC ALGORITHM FOR SEQUENCING THE MULTIOBJECTIVE FLOWSHOP

This chapter describes a new genetic algorithm that obtains Pareto optimal solutions faster than NSGA. When NSGA is applied to realistic multiobjective problems it is often seen that it lacks somewhat in both on-line performance (converging rapidly to good solutions) and off-line performance (ensuring superior quality of the final solutions). One major reason for this is that NSGA *does not* preserve the good solutions found from one generation to the next generation. Thus, good (near-optimal) solutions lost in one generation have only a probabilistic chance in NSGA to reappear in the future. Also, the *number* of final solutions on the Pareto optimal front in NSGA often remains relatively low even with good choice of parameters and even after many generations, unless large population sizes (> 250) are used.

To deal with this deficiency, an enhancement can be devised for NSGA that effects a significant performance improvement. Such a GA would be *elitist* in that it would consciously preserve a controlled fraction of the best structures or solutions present in a generation. The value of preserving the elite is well recognized. Goldberg (1989) tells us that De Jong (1975) during his Ph. D. work discovered that an elitist GA plan significantly improves both on-line and off-line performance on unimodal surfaces.

Goldberg also notes that elitism improves local search at the expense of global perspective. The enhanced version of NSGA that we describe here is called ENGA (the *Elitist Nondominated Sorting GA*).

This chapter also discusses the C++ implementation of ENGA as carried out by Jayaram (1998). We then use a multiobjective flowshop scheduling problem to illustrate the operation of ENGA.

10.1 THE ELITIST NONDominated SORTING GENETIC ALGORITHM (ENGA)

ENGA is an enhancement of NSGA but unlike NSGA, ENGA is designed so as not to mercilessly discard the old population and replace it completely by progenies. Like NSGA, ENGA uses nondominated sorting, niche formation, and sharing of fitness based on Pareto ranking. Also like NSGA, ENGA first produces the progenies through crossover and mutation (equal in number to parents). But it uses a different selection procedure. It first ranks the candidate constituents of the next generation by performing an additional nondominated sorting of the *combined parents + progenies* pool. A controlled fraction (here the top 50%) of the individuals in this combined pool is then selected to form the next generation, ready to mate and propagate their superior (nondominating) schema characteristics.

Thus each generation may end up containing several members of the *parent* chromosomes *without modification* if these parents are good enough to be able to outrank (in the nondomination sense) some of the newly-created progenies. This is akin to copying the single best solution (the "elite") of a given generation to the next generation in the elitist SGA working with a single objective (Goldberg, 1989). The skimming of the top 50% of the members of the combined (parents + progeny pool) to construct the subsequent generation (this preserves several of the previous generation's good Pareto optimal solutions "as is") makes ENGA "elitist."

By this enhancement ENGA is able to fill up the Pareto optimal front quicker than NSGA. Beyond filling up the Pareto front, in subsequent iterations it improves the solutions by the combined effects of recombination, mutation and fitness sharing.

Section 10.7 discusses the parameterization of ENGA. As discussed in Chapter 3, the method of design of experiments is used. Section 10.8 presents the results of an application of ENGA to a tri-objective flowshop sequencing problem.

Figure 10.1 outlines the logic of ENGA. The following sections describe the salient applets of ENGA as implemented in C++ for multiobjective flowshop sequencing by Jayaram (1998). The "Box #" shown in Figure 10.1 will facilitate you in linking the different steps in ENGA to the material given below.

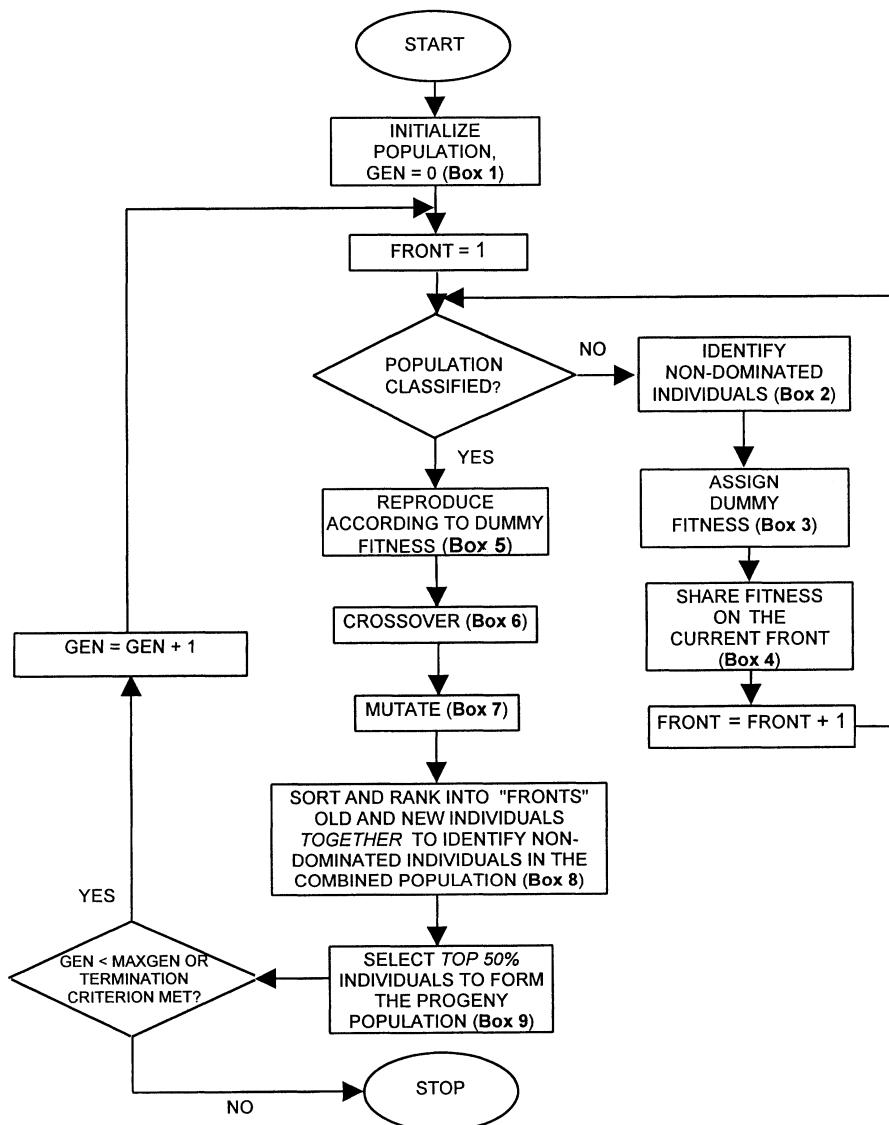


FIGURE 10.1 THE ELITIST NONDominated SORTING GA

10.2 INITIALIZATION OF ENGA (BOX 1)

The following description uses a flowshop problem with three performance objectives, minimization of makespan, minimization of mean flow time, and the minimization of the mean tardiness of jobs, to illustrate the implementation of ENGA. Like NSGA, ENGA starts with a random initial population of *job sequences*. As described in Section 4.5, these job sequences are coded as "chromosomes." Thus, together, these chromosomes constitute the population at Generation 0.

Recall that it is conventional to initiate GAs with a randomized starting population of solutions. For the flowshop scheduling problem involving n jobs, each starting sequence is random permutation of integers 1 to n with no repetition. Thus, if ten jobs are to be sequenced then the first job will be represented (identified) in this sequence of alleles by the integer "1," another job will be identified by the integer "2," and so on, up to "10."

The actual creation of the initial chromosomes thus involves the use of random integers. In the implementation being presented, the following C++ code incorporating the built-in randomizer `lrand48()` was used to generate an integer random number between two specified integers `irn1` and `irn2`:

```
int int_rnd_n1_n2(int irn1, int irn2)
{
    float irunit=1.0/(irn2-irn1+1);
    float ira=lrand48();
    float irb=ira/pow(2.0,31.0);
    float irc=irb/irunit;
    int ird=irn1-1+int(irc+1);
    if(ird<irn1 || ird>irn2)
        {cout<<"\nWrong
        intrnd("<<irn1<<,""<<irn2<<")="<<ird;exit(1);}
    else return ird;
}
```

Next, one has to ensure that no integer thus generated is repeated in the chromosome being formed. To achieve this the following code was used:

```

j=0;
while(j<JOBS)
{
x=int_rnd_n1_n2(1,JOBS);
counter=0;
if (x==oldpop[i].chrom[k])
counter=1;
if(counter==0)
{
    oldpop[i].chrom[k]=x ;
    j++;
}
}

```

In above, first a random job (identified by " x ") is chosen among the jobs numbering 1 to $JOBS$. Next one checks whether job x has already been put in the sequence (chromosome) being formed or not. If x is not already sequenced, then it is inserted in the position (allele) being currently filled in the chromosome. Otherwise, another "random job" is chosen.

The procedure is repeated till all the jobs are sequenced and a full chromosome has been constructed.

10.3 PERFORMANCE EVALUATION

In the illustration of optimally sequencing a multiobjective flowshop being presented, *three distinct and conflicting* scheduling objectives are considered. The ultimate goal is to evaluate the final quality of each chromosome (the "solution" to the flowshop scheduling problem) in the Pareto optimality sense.

The three scheduling objectives used in this implementation were respectively (1) makespan of the sequence, (2) mean flow time of the jobs, and (3) the mean tardiness of jobs.

The subroutines described in the following subsections use pre-specified job processing times and then evaluate makespan, mean flow time or mean tardiness, as relevant, for a given job sequence contained in the array `seq[JOBS]`.

10.3.1 Makespan Module

In scheduling literature, makespan is defined as the maximum completion time of all jobs, or the time taken to complete the last job on the last machine in the schedule—assuming that the processing of the first job began at time 0. Makespan is denoted by c_{\max} and computed as

$$c_{\max} = \max_{i \leq j \leq n} \{F_j\} \quad (10.1)$$

where F_j is the *flow time* for job j (the total time taken by job j from the instant of its release to the shop to the time its processing by the *last* machine is over). The following module calculates the flow times of jobs and subsequently uses (10.1) to determine the makespan for the sequence represented by `seq[JOBS]`.

```
Double makespan(int seq[JOBS])
{
int i,j;
flowtime[0][0]=processtime[0][seq[0]];
for(j=0;j<JOBS;j++)
flowtime[0][j]=flowtime[0][j-1]+
    processtime[0][seq[0]];
for(i=1;i<MACHS;i++)
{
flowtime[i][0]=flowtime[i-1][0]+
    processtime[i][seq[0]];
for(j=1;j<JOBS;j++)
{
if(flowtime[i-1][j]<=flowtime[i][j-1])
    flowtime[i][j]=flowtime[i][j-1]+
        processtime[i][seq[j]];
else
    flowtime[i][j]=flowtime[i-1][j]+
        processtime[i][seq[j]];
}
}
return (flowtime[MACHS-1][JOBS-1]);
}
```

In the above module first the flow time for each job in `seq[JOBS]` is calculated, starting with the first job, progressively moving from machine 1 to the last machine. The flow time of the last job is returned by the code as the makespan of the sequence represented by `seq[JOBS]`.

10.3.2 Mean Flow Time Module

The mean flow time measures the average response of the schedule to individual demands of jobs for service. Mathematically, mean flow time is the average of the flow times of all jobs. It is usually represented by \bar{F} and expressed as

$$\bar{F} = \frac{1}{JOBS} \sum_{j=1}^{JOBS} F_j \quad (10.2)$$

where F_j is the flow time of job j . The following code gives the mean flow time once the flow time of each job has been computed. Recall that flow times for the individual jobs were calculated in the makespan module described in the previous subsection.

```
Double meanflowtime(void)
{
float k=0;
int i;
for (i=0;i<JOBS;i++)
{
k+=flowtime[MACHS-1][i];
k/=JOBS;
return(k);
}
}
```

10.3.3 Mean Tardiness Module

The *lateness* of a job measures the conformity of the schedule to that job's due date. Lateness is defined as the amount of time by which the completion time of a job *exceeds* its due date. Mathematically

$$L_j = C_j - d_j \quad (10.3)$$

where

- L_j : Lateness of job j
- C_j : Completion time of job j
- d_j : Due date of job j

The *tardiness* (T_j) of job j is defined as the lateness L_j of job j if job j fails to meet its due date, and zero otherwise. Mathematically

$$T_j = \max\{0, L_j\} \quad (10.4)$$

Subsequently, mean tardiness (\bar{T}) is defined as the average of tardiness of all jobs. Mathematically

$$\bar{T} = \frac{1}{JOBS} \sum_{j=1}^{JOBS} T_j \quad (10.5)$$

The code below first calculates lateness of every job sequenced in `seq[JOBS]`. Then it calculates the tardiness of these jobs. The code returns the average of the sum of tardiness of all jobs as the mean tardiness of the sequence represented as `seq[JOBS]`.

```
Double meantardiness(int seq[JOBS])
{
    int i;
    double k=0;
    for (i=0;i<JOBS;i++)
        lateness[seq[i]]=flowtime[MACHS-1][i]-
                      duedate[seq[i]];
    for (i=0;i<JOBS;i++)
    {
        if(lateness[seq[i]]<=0)tardiness[seq[i]]=0;
        else tardiness[seq[i]]=lateness[seq[i]];
    }
    for (i=0;i<JOBS;i++) k+=tardiness[i];
    k/=JOBS;
    return (k);
}
```

10.4 GENETIC PROCESSING OPERATORS

The role of *genetic operators* in GA (and in ENGA) is to produce productive exchange of "genetic material" among the chromosomes to enact an intelligent search of the solution space, so as to progressively approach a global optimum. The genetic processing operators incorporated in ENGA were *selection* and *reproduction* based on nondomination, *crossover* and *job-interchange* mutation respectively.

The form of crossover employed and also the form of mutation employed, as it is now widely reported, can have a great deal of impact on a GA's overall efficacy. However, the particular crossover and mutations method that work optimally have been found to be problem domain-dependent (Starkweather, McDaniel, Mathias, Whitley and Whitley, 1991). In the experimental pilot studies done with many flowshop problems *one-point crossover* and *adjacent job-interchange* mutation performed most effectively. Consequently, these particular genetic operators were used and implemented, as described below.

10.4.1 Reproduction (Box 5)

In ENGA, as in NSGA, reproduction is performed by giving preference to *nondominated* members. Therefore, before reproduction is performed, the population is *ranked* and *classified* on the basis of each individual solution's nondomination rank. In the present implementation the C++ procedure called **front(void)** classifies the population into nondominated fronts (as done in NSGA) and assigns dummy fitness values to individual solutions according to the principle of nondomination (Section 4.5). Subsequently, procedure **phenoshare** (described in the next section) accomplishes sharing of dummy fitness values—to produce the selection metric *niche count* (Section 8.7.3).

- **The Phenotypic Sharing Module**

In ENGA, *phenotypic sharing* described by Deb and Goldberg (1989) was used to derate an individual's dummy fitness due to the presence of other individuals in its phenotypic proximity or neighbourhood.

For each nondominated individual j (indicated by `flag==2`) the metric niche count is calculated using the power-law sharing function $Sh(d)$ (Section 8.3). Distance d is then calculated by the C++ procedure `distance` given as follows:

```
double distance(double jf1, double jf2, double jf3,
double if1 double if2 double if3)
{
double d = pow((pow((jf1-if1),2) + pow((jf2-if2),2)
+ pow((jf3 - if3),2)),0.5);
return (d);
}
```

The procedure `distance` returns the phenotypic distance (d) between two individuals i and j having fitness values (based on the three optimization objectives) $jf1$, $jf2$, $jf3$ and $if1$, $if2$, $if3$ respectively. The overall C++ code for phenotypic sharing is as follows:

```
void phenoshare(void)
{
int i,j;
double d,nichecount;
for(j=0;j<popsiz; j++)
{
    nichecount=1.0;
    if(olddpop[j].flag==2)
    {
for(i=0;i<popsiz; i++)
{
    if(i==j) continue;
    if(olddpop[i].flag==2)
{d=distance(olddpop[j].fitness1, oldpop[j].fitness2,
olddpop[j].fitness3, oldpop[i].fitness1,
oldpop[i].fitness2, oldpop[i].fitness3);
if(d<0.0) d=-1.0*d;
if(d<=0.00001) nichecount++;
else if (d<dshare) nichecount+=(1-(d/dshare))*(1-
(d/dshare));
}
}
olddpop[j].dumfitness/=nichecount;
}
}
```

- **Mating Pool Formation Module**

In this module the mating pool of parent sequences is formed by the stochastic remainder selection process (Goldberg, 1989), using the "shared dummy fitness value" as the sequence's surrogate fitness for selection. The stochastic remainder process has been cited in the literature as one that performs better than the roulette wheel selection process (Deb, 1995). The following code implements this process in ENGA:

```
void preselect(void)
{
int i,jassign,k,l,winner;
double expected,num,fraction[MAXPOP];
sumfitnessdum=0;
for(l=0;l<popsizel++)
sumfitnessdum+=oldpop[i].dumfitness;
avgdum= sumfitnessdum/popsizel;
if(avgdum==0){
for(j=0;j<popsizel) choices[j]=j;
}
else {
j=0;k=0;do
{
expected=oldpop[j].dumfitness/avgdum;jassign=(int)
(expected); fraction[j]=expected-
(double)jassign;while (jassign>0)
{j assign--; choices[k]=j; k++;}j++;
}
while (j<popsizel);
j=0;
while (k<popsizel)
{ if(j>=popsizel) j=0;
if (fraction[j]>0) {
    num=real_rnd_0_1();
    if(num<fraction[j]) winner=0;
else winner=1; if(winner==0)
{choices[[k]]=j; fraction[j]-=1; k++}
}
j++;
}
}
nremain=int (popsizel)-1;
}
```

10.4.2 Crossover (Box 6)

The progeny are produced from the mating parent population by the action of several genetic operators. One method uses *recombination*, also called *crossover*. Several different ways have been proposed for implementing this operator for sequence-type chromosomes (see Murata et al, 1994, for instance). In the present implementation the *single-point crossover* operator was used. Note that in the context of sequencing, crossover is implemented significantly differently from how crossover is performed in numerical optimization by GA. In solving the scheduling problem one has to prevent creation of *infeasible* solutions (sequences) using special "repair" methods. One such scheme is as follows.

In n -job m -machine flowshop sequencing, each allele in a chromosome is a unique integer between 1 and n . Thus, obtaining feasible sequences (chromosomes) cannot be guaranteed if we arbitrarily cross, for example, two parents sequence 132456 and 251364. Suppose that we cross the genetic materials here after the third allele. Thus we might begin with parent sequences

132456 and 251364

to produce (after crossing over after the third allele) progenies

132346 and 251456

Clearly, both the progeny sequences are infeasible for a flowshop (Section 4.6.1). Progeny 132346 processes job 3 twice while job 5 is not processed at all. This demonstrates that GAs applied to scheduling must use only those crossover operators that ensure feasible solutions, or subsequently "repair" the progeny to make them feasible. This point has been noted by several earlier investigators including Starkweather et al. (1991).

The chromosome repair scheme that was implemented by Jayaram (1998) is as follows. The scheme would accept sequences 132456 and 251364 and then produce the two legitimate (feasible) progenies. First, a *single* crossover point for the two parents is chosen randomly (say the point after the third allele). Then, progeny 1 receives the first three alleles from, and in the same order as, the first three alleles from parent 1. Similarly, progeny 2 receives the first three alleles from, and in the same order as, the first three alleles in the parent 2.

Subsequently, progeny 1 is completed by placing the allele values from parent 2 (i.e. jobs) *that have not yet appeared* in progeny 1. Thus, the three remaining positions in progeny 1 would be filled by taking jobs 5, 6 and 4 in the order in which they appear in parent 2. Similarly, for progeny 2, the three remaining positions are filled by the jobs 3, 4 and 6 in the order in which they appear in parent 1. Shown below is the C++ code implementing this procedure. The scheme will always produce feasible job sequences.

```

Void crossover(int c1[JOBS], int c2[JOBS])
{
double y;
int k,f,j,p,cp,tempc1[JOBS], tempc2[JOBS];
y=int_rnd_n1_n2(1,100);
if(y<=PCROSS)
{
    cp= int_rnd_n1_n2(0,JOBS-1);
    for(k=0;k<=cp;k++) {
        tempc1[k]=c1[k];tempc2[k]=c2[k];}
    p=0;
    for(j=0;j<=JOBS;j++)
    {
        f=0;
        for(k=0;k<=JOBS;k++)
        if(c2[j]== tempc1[k]) f=1;
        if(f== 0)
        {p=p+1;tempc1[cp+p]=c2[j]; }
    }
    p=0;
    for(j=0;j<=JOBS;j++)
    {
        f=0;
        for(k=0;k<=JOBS;k++)
        if(c1[j]== tempc2[k]) f=1;
        if(f== 0)
        {p=p+1;tempc2[cp+p]=c1[j]; }
    }
}
else if (y>PCROSS) {
for(j=0;j<=JOBS;j++)
{tempc1[j]=c1[j]; tempc2[j]=c2[j];}
}
}

```

In this code first a random crossover site cp is chosen, then progeny 1 (represented by `tempc1`) receives the first cp alleles from, and in the same order as, the first cp alleles from parent 1 (represented by `c1[j]`). Similarly, progeny 2 (represented by `tempc2`) receives the first cp alleles from, and in the same order as, the first cp alleles in the parent 2 (represented by `c2[j]`). Subsequently progeny 1 (`tempc1`) is completed by placing the remaining (`JOBS - cp`) allele values (i.e. jobs) that have not yet been placed in progeny 1. This is done in the following manner. Starting from the first allele of parent 2 (`c2`), each allele in parent 2 is checked for its presence in progeny `tempc1`. If the allele is not in `tempc1`, then it is placed in progeny `tempc1`. Otherwise it is dropped and the next allele in parent 2 (`c2`) is tested for its presence in `tempc1`. The process continues till all "missing" jobs have been put in `tempc1`. In the similar manner progeny `tempc2` is constructed.

10.4.3 Mutation (Box 7)

"Mutation" is the other fundamental operator utilized in the Genetic Algorithm. Mutation helps introduce diversity in a population that may otherwise converge prematurely without exploring the solution space sufficiently. Goldberg (1989) noted that mutation is a mechanism by which *local* search in the neighbourhood of a *single* solution is performed, by introducing *small* random perturbations in the genetic makeup (genes) of that solution. For sequencing problems many variations of the mutation operator have been devised, including two adjacent-job change, arbitrary three job change, shift change, etc. However, their efficacy is problem domain-specific (Starkweather et al., 1991). For ENGA, the adjacent two-job change mutation operator has been used. This operator works as follows:

Adjacent two-job change causes two adjacent jobs in a given sequence to switch positions. Note that such mutation always produces a feasible solutions. The *pair* of adjacent jobs to change positions is chosen randomly along the sequence string, Thus,

132456 sequence before mutation

134256 after two-adjacent-job-change mutation

The C++ code implementing this mutation is shown below:

```
y=real_rnd_0_1( ) *100;
if(y<=PMUTE)
{
    za1= int_rnd_n1_n2(0,JOBS-2);
//Swap allele za1 with za1+1
    {
        temp=tempc1[za1];
        tempc1[za1]= tempc1[za1+1];
        tempc1[za1+1]=temp;
    }
}
```

In this code first mutation site za1 is randomly selected in progeny tempc1. Then allele tempc1[za1] is swapped with allele tempc1[za1+1] in progeny tempc1.

10.5 THE ADDITIONAL NONDOMINATED SORTING AND RANKING STEP IN ENGA (Box 8 and Box 9)

As mentioned at the beginning of this chapter, ENGA does not mercilessly discard the old population and replace it completely by its progeny. Rather, ENGA performs an *additional* nondominated sorting on the combined pool consisting of members of old population and its progeny and then selects the top 50% individuals to constitute the next generation. In the present C++ implementation, procedure **frontelite** performs the classification of combined population into nondominated fronts.

In this procedure first all the members of the population are assigned a flag value equal to 0 signifying that these members are to be classified. Then each individual is compared with other individuals of the population on the basis of nondomination described in Section 4.5.

If a member is not dominated by any other individual in the population, it is assigned a flag value of 2 (signifying *nondominated*). Otherwise the individual is assigned a flag value of 1 (signifying *dominated*). All individuals having a flag value of 2 are assigned a *front number* = 1 (signifying its residence on the *Pareto-optimal front*).

The procedure is repeated leaving the already classified members (now assigned a flag value of 3) aside till the whole population is classified into fronts.

The C++ code for this procedure is shown below.

```
Void frontelite(void)
{
int i,j,front_index,popcount;
front_index=1;popcount=0;
while(popcount<2*popszie)
{
    for(j=0;j<2*popszie;j++)
    {
        if(elite[j].flag==3) continue;
        for(i=0;i<2*popszie;i++)
        {
            if (i==j) continue;
            else if(elite[i].flag==3) continue;
            else if(elite[i].flag==1) continue;
            if(elite[j].fitness1>elite[i].fitness1
               && elite[j].fitness2>elite[i].fitness2
               && elite[j].fitness3>elite[i].fitness3)
                elite[j].flag=1;
        }
        if (elite[j].flag==0)
            {elite[j].flag=2;popcount++}
    }
    for(i=0;i<2*popszie;i++)
    {
        if (elite[i].flag==2) elite[i].front=front_index;
        front_index++;    }
        for(i=0;i<2*popszie;i++)
        { if (elite[i].flag==2) elite[i].flag=3;
        else if (elite[i].flag==1) elite[i].flag=0; }
}
```

After classification is complete, the population is sorted on the basis of front numbers. The procedure **sort** in the ENGA implementation uses the bubble sort method. The top 50% of sorted population constitute the members of the next generation.

10.6 STOPPING CONDITION AND OUTPUT MODULE

The stopping condition for ENGA implementation is the *maximum number of generations* to be executed, a quantity pre-set by the analyst. The output routine first prints the values of ENGA parameters p_s , p_c , p_m and σ_{share} . Then it prints the solution sequences and their three objective function values (makespan, mean flowtime and mean tardiness), along with the front (indicating whether the solution is Pareto-optimal or not). Results display the members of the population at each generation. The Pareto-optimal solution sequences are put in a separate file.

10.7 PARAMETERIZATION OF ENGA BY DESIGN OF EXPERIMENTS

As we pointed out in Chapter 3, a critical difficulty in applying GAs is that the various parameters must be correctly chosen to ensure the GA's satisfactory on-line and off-line performance. Using ENGA satisfactorily would be no exception. Additionally, as mentioned in Chapter 3, the answers found here (the optimum values for the GA parameters p_s , p_c , p_m , etc.) are often problem-dependent. Further, Davis (1991) notes that crossover and mutation effects can interact and "support each other in important ways" and observes that a judicious blend of mutation and crossover does better than either one alone to strike a good balance between exploration of the total solution space and exploitation of good solutions currently at hand. It is easy to see, therefore, that the optimization of different GA parameters itself is a global search problem and one that must be tackled in the problem domain of interest, before we apply the GA in a "production run." However, in some GA applications only modest effort is extended to optimizing the parameters. In other situations exhaustive search is resorted to (see, for instance, Murata, Ishibuchi and Tanaka, 1996). While the first approach exposes one to the risk of the GA performing badly, exhaustive search is perhaps wasteful of effort.

Since the GA process is controlled by multiple factors whose effects may interact and also that it is a process without a known response structure, one may attempt parameterization using a design-of-experiments (DOE) framework, as described in Chapter 3. The object here will be

1. To identify parameters which have significant effect on the convergence of the GA being implemented, and
2. To determine the levels (values) for the relevant parameters in order that their effects *speed up* the convergence of the GA.

It must be understood here that the aim is to experimentally seek good parameter values, and not to find the *exact* cause-effect relationships between the parameters and the GA's response. Therefore, the factors may be examined even at only two levels and a partial factorial (orthogonal array) design may be used (Montgomery, 1996). Clearly, this would not allow a study of the non-linear influence of parameters or the effect of three-or four-factor interactions. Also, no interpolation or extrapolation would be attempted. Subsequently, to yield a better understanding of effects of each parameters on the algorithm, experiments with factors at more than two levels and a full factorial design study could be attempted, as done for instance by Prasad (1997).

10.7.1 The Response to be Observed

When we use SGA, we are interested in finding the single best solution. The parameter experimentation responses chosen may then be the *best*, *average*, and the *worst* value of the objective function found after executing a fixed number of GA iterations or function evaluations. But in multiobjective optimization, which is the domain of ENGA, we would be interested in a good *set* of Pareto-optimal solutions. Good Pareto optimal solutions have two properties: (1) They lie on the Pareto-optimal front, and (2) They are well-dispersed on the front (the solutions should not form "clumps").

To parameterize ENGA, the concept of *nondominance* was used to select the response factor, as follows. All solutions obtained in a fixed number of GA iterations by conducting a set of DOE experiments were pooled together and then subjected to non-dominated sorting. The *count* of *distinct Front 1 members* being contributed by each experiment in the total DOE set was chosen as the response. Dispersion was evaluated subjectively.

The ENGA process, like any GA process, is a directed search with stochastic sampling occurring in each generation. The effectiveness of

this search is observed to be affected by the mutation and crossover probabilities p_m and p_c , the population size p_s , and the ENGA parameter σ_{share} , which controls the precision with which the Pareto optimal front is combed for the existence of a possible optimal point or peak. We use here the 15 machine-49 jobs flowshop problem described in Section 4.4 to illustrate the procedure, the flowshop sequencing objectives being minimization of *makespan*, minimization of *mean flowtime*, and minimization of *mean tardiness*. The processing times and due dates for the jobs in this problem are shown in Table 4.1.

10.7.2 The Parameterization Experiments

The initial experimental layout was 2^4 full factorial involving four factors, namely, population size p_s , the probability of crossover p_c , the probability of mutation p_m and the ENGA fitness sharing parameter σ_{share} with two possible levels for each parameter (judged to be possibly correct). To remove the bias possible due to the starting GA sequences and also to assure robustness in the GA's performance, three randomly produced initial starting (job) sequences were tested in each experiment, providing the necessary replications. Furthermore, these three random starting sequences were kept unaltered in each of the 16 full factorial experiments to involve the principle of variance reduction (Law and Kelton, 1991). After this initial DOE run was completed, it was found that high values of crossover probability performed consistently better regardless of the settings of the other parameters. Subsequently 2^3 full factorial experiments were conducted with five randomly produced initial sequences giving five replications for each run. Crossover probability was kept fixed at 0.9.

Table 10.1 shows the two levels of the factors used in the 2^3 DOE run. Each experiment was run for 200 generations assuming that GA would then get sufficiently close to the true though unknown Pareto-optimal solutions. After the runs terminated, the results were pooled together and subjected to non-dominated sorting, to appraise which particular parameter combination contributed the *maximum number of unique solutions* to the Pareto optimal front (the metric selected to be the response measure for the DOE runs).

Table 10.2 shows the 2^3 full factorial experimental set-up with the summary of the results obtained. Figure 10.2 displays the factor effects and the interaction between the parameters, indicating the relatively strong impact on convergence of solutions to the Pareto front of population size (p_s), probability of mutation (p_m) and interaction between p_s and p_m .

The above results also indicated that a *high* population size (p_s), *low* σ_{share} (ds), *low* probability of mutation (p_m) and a *high* probability of crossover (p_c) would be the best parameter combination for ENGA. Subsequently, in implementation of ENGA and similarly for NSGA, these parameters would be set at the following values:

Population size (p_s) = 200, σ_{share} (ds) = 1,
 probability of crossover (p_c) = 0.9 and
 probability of mutation (p_m) = 0.0.

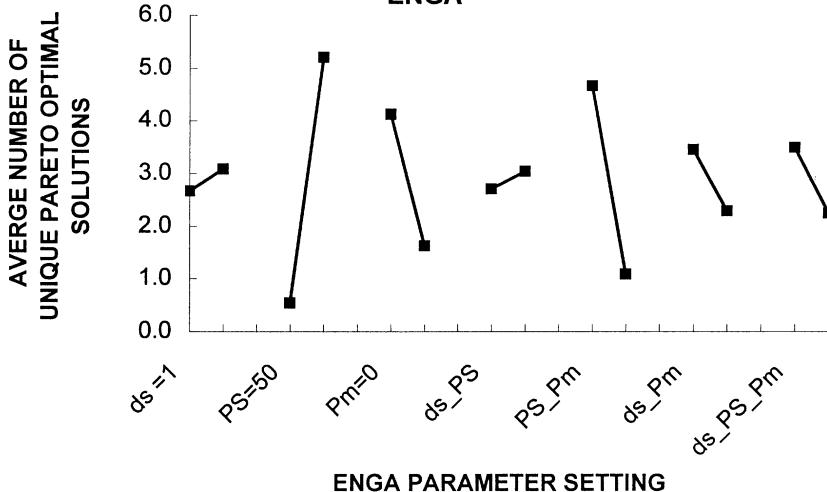
TABLE 10.1 DOE FACTOR LEVELS

Factors	Level 1	Level 2
σ_{share} (ds)	5	1
Population size	100	50
Probability of mutation	0.01	0.00

TABLE 10.2 RESULTS OF THE 2^3 FULL FACTORIAL ENGA PARAMETERIZATION EXPERIMENT

Expt #	Factors			Random Seed used						Average No. Of Unique Pareto Solutions
	ds	p_s	p_m	#1	#2	#3	#4	#5	#6	
1	1	50	0	0	0	0	0	0	0	0.00
2	1	50	0.01	0	0	0	0	0	6	1.00
3	1	100	0	11	6	0	0	23	0	6.67
4	1	100	0.01	0	0	1	2	8	7	3.00
5	5	50	0	0	0	0	0	0	0	0.00
6	5	50	0.01	0	0	0	0	7	0	1.17
7	5	100	0	0	0	28	13	9	9	9.83
8	5	100	0.01	0	4	0	4	0	0	1.33

FIGURE 10.2 FACTOR EFFECTS ON THE DISCOVERY OF PARETO OPTIMAL SOLUTIONS BY ENGA



10.7.3 Concluding Remarks on Parameterization

The above example has illustrated the methodology of statistical design of experiments to parameterize a multiobjective genetic algorithm (ENGA). It has used a *special* response function (the count of solutions contributed by to front1) by the different possible parameter settings emanating from the experimental design. This methodology has been tested on a variety of different flow, job and open shop scheduling problems as test platforms.

DOE-based parameterization appears to work considerably better than the use of ad hoc p_s , σ_{share} (ds), p_m or p_c values arbitrary "rules-of-thumb" parameterization guidelines prescribed variously in the GA literature. It has been widely reported that the same "optimum" GA parameters do not work on all problems because it appears that there exists considerable interaction because of the particular stochastic process that ensues as the GA executes.

In this context we note that even though the DOE method is exploratory, it gives the analyst freedom to make parameterization *problem-specific*.

10.8 APPLICATION OF ENGA TO THE 49-JOB 15-MACHINE FLOWSHOP

Among many flowshop problems tested, we used the 49-job 15-machine problem shown in Table 4.1 solved earlier by NSGA (Section 9.5). Table 10.3 displays sample tri-objective Pareto optimal solutions to this problem found by a typical 250-generation run of ENGA. Figure 10.3 shows the relative position of these solutions graphically. Table 10.4 shows the number of Pareto solutions found in thirty seven different ENGA trials, executing up to 250 generations. A comparison of this data with Table 9.1 hints that inclusion of elitism improves the performance of nondominated sorting GAs. We return to this comparison in Section 11.3 of the next chapter.

10.9 CHAPTER SUMMARY

This chapter introduced the Elitist Nondominated Sorting Genetic Algorithm (ENGA), an enhancement to NSGA, a genetic algorithm that rapidly discovers Pareto optimal solutions to multiobjective problems.

ENGA works similar to NSGA in that it uses niche formation and sharing based on Pareto ranking. Like NSGA, ENGA produces the progenies through crossover and mutation (equal in number to parents), and then does selection. But in selection, unlike NSGA, ENGA is designed so as not to mercilessly discard the old population and replace it completely by progenies.

ENGA uses the following selection procedure. It first ranks the candidate constituents of the next generation by performing an additional nondominated sorting of the combined *parents + progenies* pool. Then a controlled fraction (here the top 50%) of the solutions in this combined pool is selected to form the next generation, ready to mate and propagate their superior (nondominating) schema characteristics.

In this chapter we also described the steps involved in implementing ENGA in C++, using multiobjective flowshop scheduling as an illustration. In the next chapter we conduct a computational study of the relative performance of NSGA and ENGA in solving a random assortment of multiobjective flowshop problems.

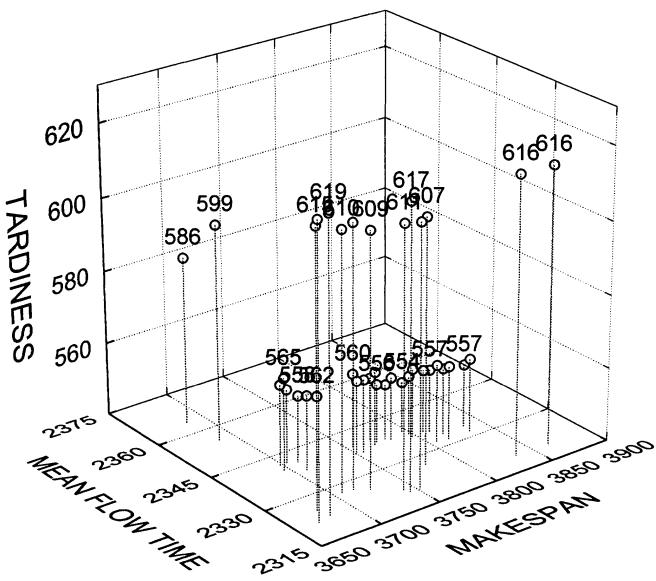


FIGURE 10.3 TRI-OBJECTIVE PARETO OPTIMAL SOLUTIONS FOUND BY ENGA FOR THE 49-JOB 15-m/c FLOWSHOP

TABLE 10.3 SAMPLE PARETO OPTIMAL JOB SEQUENCES FOR THE TRI-OBJECTIVE 49-JOB 15-m/c FLOWSHOP OF TABLE 4.1 FOUND BY 250 GENERATIONS OF ENGA

OPTIMAL JOB SEQUENCE													MAKESPAN	MEAN FLOW TIME	TARDI-NESS
8	3	16	17	25	21	24	9	11	22	7	23		3778	2336.57	558
2	29	32	18	26	47	6	14	5	13	43	27				
10	34	1	31	28	36	30	12	41	46	42	44				
33	20	39	4	35	37	38	40	19	45	49	15				
48															
8	3	16	17	25	21	24	9	11	22	7	23		3755	2339.55	560
2	29	32	18	14	5	26	47	6	13	43	27				
10	34	1	31	28	36	30	12	41	46	42	44				
33	4	39	37	20	38	35	49	40	19	15	48				
45															
8	3	16	17	25	21	24	9	11	22	7	23		3707	2335.47	562
2	29	32	18	26	47	6	14	5	13	43	27				
10	34	1	31	28	36	30	12	41	46	42	44				
33	20	39	4	35	40	49	19	38	15	48	45				
37															
8	3	16	17	25	21	24	9	11	22	26	47		3685	2363.88	586
7	6	23	2	29	32	18	14	5	13	43	27				
10	34	1	31	28	36	30	12	41	46	42	44				
33	20	39	4	35	40	48	37	38	19	45	49				
15															

TABLE 10.3 CONTD. SAMPLE PARETO OPTIMAL JOB SEQUENCES FOR THE 49-JOB 15-m/c FLOWSHOP FOUND BY ENGA

OPTIMAL JOB SEQUENCE														MAKESPAN	MEAN FLOW TIME	TARDI-NES
8	3	16	17	25	21	24	9	11	22	7	23			3795	2338.47	554
2	29	32	18	14	5	26	47	6	13	43	27					
10	34	1	31	28	36	30	12	41	46	42	44					
33	4	39	20	35	37	38	40	19	45	49	15					
48																
8	3	16	17	25	21	24	9	11	22	7	23			3780	2338.45	555
2	29	32	18	14	5	26	47	6	13	43	27					
10	34	1	31	28	36	30	12	41	46	42	44					
33	20	39	4	35	37	38	40	19	45	49	15					
48																
8	3	16	17	25	21	24	9	11	22	7	23			3798	2338.22	555
2	29	32	18	14	5	26	47	6	13	43	27					
10	34	1	31	28	36	30	12	41	46	42	44					
33	4	39	20	35	37	38	40	49	19	45	15					
48																
8	3	16	17	25	21	24	9	11	22	7	23			3772	2338.29	556
2	29	32	18	14	5	26	47	6	13	43	27					
10	34	1	31	28	36	30	12	41	46	42	44					
33	4	39	37	20	38	40	19	45	35	49	15					
48																
8	3	16	17	25	21	24	9	11	22	7	23			3798	2337.51	556
2	29	32	18	14	5	26	47	6	13	43	27					
10	34	1	31	28	36	30	12	41	46	42	44					
33	4	39	20	35	37	38	49	40	19	45	15					
48																
8	3	16	17	25	21	24	9	11	22	7	23			3783	2337.78	557
2	29	32	18	14	5	26	47	6	13	43	27					
10	34	1	31	28	36	30	12	41	46	42	44					
33	20	39	4	35	37	38	49	40	19	45	15					
48																
8	3	16	17	25	21	24	9	11	22	7	23			3777	2337.82	557
2	29	32	18	14	5	26	47	6	13	43	27					
10	34	1	31	28	36	30	12	41	46	42	44					
33	4	39	37	20	38	35	49	40	19	45	15					
48																
8	3	16	17	25	21	24	9	11	22	7	23			3793	2336.92	557
2	29	32	18	26	47	6	14	5	13	43	27					
10	34	1	31	28	36	30	12	41	46	42	44					
33	4	39	20	35	37	38	40	19	45	49	15					
48																
8	3	16	17	25	21	24	9	11	22	7	23			3812	2336.14	557
2	29	32	18	14	5	26	47	6	13	43	27					
10	34	1	31	28	36	30	12	41	46	42	44					
33	4	39	20	35	40	49	19	45	38	15	37					
48																
8	3	16	17	25	21	24	9	11	22	7	23			3771	2339.82	557
2	29	32	18	14	5	26	47	6	13	43	27					
10	34	1	31	28	36	30	12	41	46	42	44					
33	20	39	4	40	37	38	35	49	19	45	15					
48																
8	3	16	17	25	21	24	9	11	22	7	23			3836	2334.08	557
2	29	32	18	14	5	26	47	6	13	43	27					
10	34	1	31	28	36	30	12	41	46	42	44					
33	20	39	4	35	49	19	45	38	40	15	37					
48																
8	3	16	17	25	21	24	9	11	22	7	23			3808	2336.67	557
2	29	32	18	14	5	26	47	6	13	43	27					
10	34	1	31	28	36	30	12	41	46	42	44					
33	20	39	4	35	40	49	19	45	38	15	37					
48																

TABLE 10.3 CONTD. SAMPLE PARETO OPTIMAL JOB SEQUENCES FOR THE 49-JOB 15-m/c FLOWSHOP FOUND BY ENGA

OPTIMAL JOB SEQUENCE													MAKESPAN	MEAN FLOW TIME	TARDI-NES
8	3	16	17	25	21	24	9	11	22	7	23		3817	2334.02	558
2	29	32	18	14	5	26	47	6	13	43	27				
10	34	1	31	28	36	30	12	41	46	42	44				
33	20	39	4	35	49	19	45	38	15	37	40				
48															
8	3	16	17	25	21	24	9	11	22	7	23		3808	2335.43	558
2	29	32	18	14	5	26	47	6	13	43	27				
10	34	1	31	28	36	30	12	41	46	42	44				
33	20	39	4	35	49	40	19	45	38	15	37				
48															
8	3	16	17	25	21	24	9	11	22	7	23		3770	2337.20	558
2	29	32	18	26	47	6	14	5	13	43	27				
10	34	1	31	28	36	30	12	41	46	42	44				
33	4	39	37	20	38	35	40	19	45	49	15				
48															
8	3	16	17	25	21	24	9	11	22	7	23		3808	2334.45	558
2	29	32	18	14	5	26	47	6	13	43	27				
10	34	1	31	28	36	30	12	41	46	42	44				
33	20	39	4	35	49	19	45	40	38	15	37				
48															
8	3	16	17	25	21	24	9	11	22	7	23		3770	2336.65	558
2	29	32	18	26	47	6	14	5	13	43	27				
10	34	1	31	28	36	30	12	41	46	42	44				
33	4	39	37	20	38	40	19	45	35	49	15				
48															
8	3	16	17	25	21	24	9	11	22	7	23		3711	2340.67	558
2	29	32	18	14	5	26	47	6	13	43	27				
10	34	1	31	28	36	30	12	41	46	42	44				
33	4	39	37	20	38	40	19	45	35	48	45				
15															
8	3	16	17	25	21	24	9	11	22	7	23		3796	2335.96	559
2	29	32	18	26	47	6	14	5	13	43	27				
10	34	1	31	28	36	30	12	41	46	42	44				
33	4	39	20	35	37	38	49	40	19	45	15				
48															
8	3	16	17	25	21	24	9	11	22	7	23		3781	2335.82	559
2	29	32	18	26	47	6	14	5	13	43	27				
10	34	1	31	28	36	30	12	41	46	42	44				
33	20	39	4	35	37	38	49	40	19	45	15				
48															
8	3	16	17	25	21	24	9	11	22	7	23		3775	2336.18	559
2	29	32	18	26	47	6	14	5	13	43	27				
10	34	1	31	28	36	30	12	41	46	42	44				
33	4	39	37	20	38	35	49	40	19	45	15				
48															
8	3	16	17	25	21	24	9	11	22	7	23		3834	2331.96	560
2	29	32	18	26	47	6	14	5	13	43	27				
10	34	1	31	28	36	30	12	41	46	42	44				
33	20	39	4	35	49	19	45	38	40	15	37				
48															
8	3	16	17	25	21	24	9	11	22	7	23		3806	2334.47	560
2	29	32	18	26	47	6	14	5	13	43	27				
10	34	1	31	28	36	30	12	41	46	42	44				
33	20	39	4	35	40	49	19	45	38	15	37				
48															
8	3	16	17	25	21	24	9	11	22	7	23		3704	2341.55	560
2	29	32	18	14	5	26	47	6	13	43	27				
10	34	1	31	28	36	30	12	41	46	42	44				
33	4	39	37	20	38	40	19	48	45	35	49				
15															

TABLE 10.3 CONTD. SAMPLE PARETO OPTIMAL JOB SEQUENCES FOR THE 49-JOB 15-m/c FLOWSHOP FOUND BY ENGA

OPTIMAL JOB SEQUENCE														MAKESPAN	MEAN FLOW TIME	TARDINESS
8	3	16	17	25	21	24	9	11	22	7	23			3769	2337.86	560
2	29	32	18	26	47	6	14	5	13	43	27					
10	34	1	31	28	36	30	12	41	46	42	44					
33	20	39	4	40	37	38	35	49	19	45	15					
48																
8	3	16	17	25	21	24	9	11	22	7	23			3709	2337.55	560
2	29	32	18	14	5	26	47	6	13	43	27					
10	34	1	31	28	36	30	12	41	46	42	44					
33	20	39	4	35	40	49	19	38	15	48	45					
37																
8	3	16	17	25	21	24	9	11	22	7	23			3815	2331.90	560
2	29	32	18	26	47	6	14	5	13	43	27					
10	34	1	31	28	36	30	12	41	46	42	44					
33	20	39	4	35	49	19	45	38	15	37	40					
48																
8	3	16	17	25	21	24	9	11	22	7	23			3764	2336.71	560
2	29	32	18	14	5	26	47	6	13	43	27					
10	34	1	31	28	36	30	12	41	46	42	44					
33	4	39	20	35	40	49	19	45	38	15	48					
37																
8	3	16	17	25	21	24	9	11	22	7	23			3806	2333.22	560
2	29	32	18	26	47	6	14	5	13	43	27					
10	34	1	31	28	36	30	12	41	46	42	44					
33	20	39	4	35	49	40	19	45	38	15	37					
48																
8	3	16	17	25	21	24	9	11	22	7	23			3755	2336.61	560
2	29	32	18	14	5	26	47	6	13	43	27					
10	34	1	31	28	36	30	12	41	46	42	44					
33	20	39	4	35	49	40	19	38	45	15	48					
37																
8	3	16	17	25	21	24	9	11	22	7	23			3764	2334.20	560
2	29	32	18	14	5	26	47	6	13	43	27					
10	34	1	31	28	36	30	12	41	46	42	44					
33	20	39	4	35	49	19	45	38	40	15	48					
37																
8	3	16	17	25	21	24	9	11	22	7	23			3750	2336.94	560
2	29	32	18	14	5	26	47	6	13	43	27					
10	34	1	31	28	36	30	12	41	46	42	44					
33	20	39	4	35	40	49	19	45	38	15	48					
37																
8	3	16	17	25	21	24	9	11	22	7	23			3709	2336.31	560
2	29	32	18	14	5	26	47	6	13	43	27					
10	34	1	31	28	36	30	12	41	46	42	44					
33	20	39	4	35	49	40	19	38	15	48	45					
37																
8	3	16	17	25	21	24	9	11	22	7	23			3756	2335.16	560
2	29	32	18	14	5	26	47	6	13	43	27					
10	34	1	31	28	36	30	12	41	46	42	44					
33	4	39	20	35	49	40	19	45	38	15	48					
37																
8	3	16	17	25	21	24	9	11	22	7	23			3750	2335.69	560
2	29	32	18	14	5	26	47	6	13	43	27					
10	34	1	31	28	36	30	12	41	46	42	44					
33	20	39	4	35	49	40	19	45	38	15	48					
37																
8	3	16	17	25	21	24	9	11	22	7	23			3806	2332.41	561
2	29	32	18	26	47	6	14	5	13	43	27					
10	34	1	31	28	36	30	12	41	46	42	44					
33	20	39	4	35	49	19	45	40	38	15	37					
48																

TABLE 10.3 CONTD. SAMPLE PARETO OPTIMAL JOB SEQUENCES FOR THE 49-JOB 15-m/c FLOWSHOP FOUND BY ENGA

OPTIMAL JOB SEQUENCE															MAKESPAN	MEAN FLOW TIME	TARDI-NES
8	3	16	17	25	21	24	9	11	22	7	23				3702	2339.92	562
2	29	32	18	26	47	6	14	5	13	43	27						
10	34	1	31	28	36	30	12	41	46	42	44						
33	4	39	37	20	38	40	19	48	45	35	49						
15																	
8	3	16	17	25	21	24	9	11	22	7	23				3753	2334.49	562
2	29	32	18	26	47	6	14	5	13	43	27						
10	34	1	31	28	36	30	12	41	46	42	44						
33	20	39	4	35	49	40	19	38	45	15	48						
37																	
8	3	16	17	25	21	24	9	11	22	7	23				3748	2334.82	562
2	29	32	18	26	47	6	14	5	13	43	27						
10	34	1	31	28	36	30	12	41	46	42	44						
33	20	39	4	35	49	40	19	45	38	15	48						
37																	
8	3	16	17	25	21	24	9	11	22	7	23				3762	2332.08	562
2	29	32	18	26	47	6	14	5	13	43	27						
10	34	1	31	28	36	30	12	41	46	42	44						
33	20	39	4	35	49	19	45	38	40	15	48						
37																	
8	3	16	17	25	21	24	9	11	22	7	23				3707	2334.22	562
2	29	32	18	26	47	6	14	5	13	43	27						
10	34	1	31	28	36	30	12	41	46	42	44						
33	20	39	4	35	49	40	19	38	15	48	45						
37																	
8	3	16	17	25	21	24	9	11	22	7	23				3698	2341.39	562
2	29	32	18	14	5	26	47	6	13	43	27						
10	34	1	31	28	36	30	12	41	46	42	44						
33	20	39	4	35	49	40	19	38	19	45	49						
15																	
8	3	16	17	25	21	24	9	11	22	7	23				3748	2333.57	563
2	29	32	18	26	47	6	14	5	13	43	27						
10	34	1	31	28	36	30	12	41	46	42	44						
33	20	39	4	35	49	40	19	45	38	15	48						
37																	
8	3	16	17	25	21	24	9	11	22	7	23				3692	2340.39	563
2	29	32	18	14	5	26	47	6	13	43	27						
10	34	1	31	28	36	30	12	41	46	42	44						
33	20	39	4	40	48	37	38	19	45	35	49						
15																	
8	3	16	17	25	21	24	9	11	22	7	23				3692	2340.06	564
2	29	32	18	14	5	26	47	6	13	43	27						
10	34	1	31	28	36	30	12	41	46	42	44						
33	20	39	4	40	48	37	38	35	49	19	45						
15																	
8	3	16	17	25	21	24	9	11	22	7	23				3696	2339.67	565
2	29	32	18	26	47	6	14	5	13	43	27						
10	34	1	31	28	36	30	12	41	46	42	44						
33	20	39	4	35	40	48	37	38	19	45	49						
15																	
8	3	16	17	25	21	24	9	11	22	7	23				3690	2338.76	566
2	29	32	18	26	47	6	14	5	13	43	27						
10	34	1	31	28	36	30	12	41	46	42	44						
33	20	39	4	40	48	37	38	19	45	35	49						
15																	
8	3	16	17	25	21	24	9	11	22	7	23				3690	2338.43	567
2	29	32	18	26	47	6	14	5	13	43	27						
10	34	1	31	28	36	30	12	41	46	42	44						
33	20	39	4	40	48	37	38	35	49	19	45						
15																	

TABLE 10.3 CONTD. SAMPLE PARETO OPTIMAL JOB SEQUENCES FOR THE 49-JOB 15-m/c FLOWSHOP FOUND BY ENGA

OPTIMAL JOB SEQUENCE															MAKESPAN	MEAN FLOW TIME	TARDI-NES
8	3	16	17	25	21	24	9	11	22	26	47				3688	2355.43	599
7	6	44	23	2	29	32	18	14	5	13	43						
27	10	34	1	31	28	36	30	12	41	46	42						
33	20	39	4	40	48	37	38	19	45	35	49						
15																	
8	3	16	17	25	21	24	9	11	22	26	47				3777	2326.73	607
7	6	44	23	42	28	12	29	36	46	33	18						
1	31	2	4	13	32	14	5	43	27	10	34						
39	37	30	38	41	20	35	40	19	45	49	15	48					
8	3	16	17	25	21	24	9	11	22	26	47				3767	2324.29	608
7	6	44	23	42	28	12	29	36	46	33	18						
1	31	2	4	13	32	14	5	43	27	10	34						
39	37	30	38	41	20	35	49	19	45	40	15	48					
8	3	16	17	25	21	24	9	11	22	26	47				3725	2325.49	609
7	6	44	23	42	28	12	29	36	46	33	18						
1	31	2	4	13	32	14	5	43	27	10	34						
39	37	30	38	41	19	40	20	45	35	48	49	15					
8	3	16	17	25	21	24	9	11	22	26	47				3707	2327.39	610
7	6	44	23	42	28	12	29	36	46	33	18						
1	31	2	4	13	32	14	5	43	27	10	34						
39	37	30	38	41	19	40	20	48	45	35	49	15					
8	3	16	17	25	21	24	9	11	22	26	47				3716	2327.22	611
7	6	44	23	42	28	12	29	36	46	33	18						
1	31	2	4	13	32	14	5	43	27	10	34						
39	37	30	38	41	20	35	40	48	19	45	49	15					
8	3	16	17	25	21	24	9	11	22	26	47				3707	2326.24	611
7	6	44	23	42	28	12	29	36	46	33	18						
1	31	2	4	13	32	14	5	43	27	10	34						
39	37	30	38	41	19	40	20	48	45	35	49	15					
8	3	16	17	25	21	24	9	11	22	26	47				3707	2326.24	611
7	6	44	23	42	28	12	29	36	46	33	18						
1	31	2	4	13	32	14	5	43	27	10	34						
39	37	30	38	41	20	40	48	19	45	35	49	15					
8	3	16	17	25	21	24	22	26	47	7	6				3752	2320.69	611
44	9	11	42	23	28	12	29	36	46	33	18						
1	31	2	4	13	14	5	32	43	27	10	34						
39	37	30	38	41	20	35	49	40	19	45	15	48					
8	3	16	17	25	21	24	22	26	47	7	6				3742	2321.94	611
44	9	11	42	23	28	12	29	36	46	33	18						
1	31	2	4	13	32	14	5	43	27	10	34						
39	37	30	38	41	20	40	19	45	35	49	15	48					
8	3	16	17	25	21	24	22	26	47	7	6				3759	2320.53	612
44	9	11	42	23	28	12	29	36	46	33	18						
1	31	2	4	13	32	14	5	43	27	10	34						
39	37	30	38	41	20	35	49	19	45	40	15	48					
8	3	16	17	25	21	24	22	26	47	7	6				3677	2324.67	615
7	6	44	23	42	28	12	29	36	46	33	18						
1	31	2	4	13	32	14	5	43	27	10	34						
39	37	30	38	41	20	35	49	19	45	40	15	48					
8	3	16	17	25	21	24	35	49	19	45	15				3699	2322.16	616
44	9	11	42	23	28	12	29	36	46	33	18						
1	31	2	4	13	32	14	5	43	27	10	34						
39	37	30	38	41	20	40	48	35	49	19	45	15					
8	3	16	17	25	21	24	22	26	47	7	6				3863	2319.73	616
44	9	11	42	23	28	12	29	36	46	33	18						
1	31	2	4	13	32	14	5	43	27	10	34						
30	41	20	39	40	48	37	38	35	49	19	45	15					
8	3	16	17	25	21	24	22	26	47	7	6				3717	2321.73	613
44	9	11	42	23	28	12	29	36	46	33	18						
1	31	2	4	13	32	14	5	43	27	10	34						
39	37	30	38	41	20	40	19	45	35	48	49	15					

**TABLE 10.4 AVERAGE NUMBER OF PARETO OPTIMAL
SOLUTIONS FOUND BY ENGA FOR THE 49-JOB 15-MACHINE
PROBLEM OF TABLE 4.1.
(Thirty seven different starting populations were used.)**

Generation →	0	25	50	75	100	125	150	175	200	225	250
Seed # 1	7	26	56	53	59	64	56	59	58	59	60
Seed # 2	2	49	69	72	81	86	92	89	95	86	86
Seed # 3	3	23	41	56	46	67	76	70	70	69	69
Seed # 4	2	14	30	33	36	36	35	35	34	33	32
Seed # 5	7	42	83	88	69	71	68	64	62	72	66
Seed # 6	8	21	42	44	51	48	51	51	49	50	49
Seed # 7	11	15	27	30	31	30	31	31	30	30	31
Seed # 8	7	32	34	31	34	35	40	32	32	33	33
Seed # 9	12	41	96	74	81	76	71	74	70	72	70
Seed # 10	5	28	55	53	53	56	57	62	66	65	66
Seed # 11	5	31	31	29	32	32	33	37	36	36	36
Seed # 12	3	33	22	26	27	25	33	38	39	40	40
Seed # 13	8	36	90	98	104	92	97	93	77	81	83
Seed # 14	7	26	96	89	88	88	88	87	88	91	97
Seed # 15	6	63	51	60	62	59	58	54	55	57	58
Seed # 16	7	14	50	35	47	45	47	45	47	45	44
Seed # 17	5	19	33	39	49	60	60	63	57	57	52
Seed # 18	11	32	54	52	56	58	57	59	54	56	59
Seed # 19	7	49	91	81	79	83	84	86	81	73	77
Seed # 20	4	13	42	46	51	53	53	52	52	49	52
Seed # 21	7	17	32	47	37	38	41	40	36	38	39
Seed # 22	9	28	23	24	32	41	41	41	42	42	42
Seed # 23	9	13	31	42	43	47	46	47	46	46	44
Seed # 24	3	47	50	68	64	56	62	61	62	64	64
Seed # 25	10	31	23	30	36	35	33	35	40	41	40
Seed # 26	5	8	31	31	32	32	32	32	32	31	32
Seed # 27	6	50	98	111	96	96	94	96	98	96	99
Seed # 28	7	43	37	40	40	39	39	39	38	39	39
Seed # 29	4	24	44	39	40	39	39	40	40	40	40
Seed # 30	6	12	43	48	49	50	48	48	46	47	46
Seed # 31	3	12	16	18	18	18	18	18	18	18	18
Seed # 32	5	27	23	33	43	26	26	26	26	26	26
Seed # 33	4	20	48	59	57	43	51	48	52	52	57
Seed # 34	4	29	37	38	36	37	37	37	37	36	37
Seed # 35	9	29	72	76	75	73	79	94	84	90	79
Seed # 36	7	13	22	15	15	15	16	16	16	16	16
Seed # 37	4	66	70	68	119	117	117	107	115	115	117
Average Count	6.2	29.1	48.5	50.7	53.2	53.1	54.2	54.2	53.5	53.8	53.9

A COMPARISON OF MULTIOBJECTIVE FLOWSHOP SEQUENCING BY NSGA AND ENGA

This chapter presents detailed results of scheduling typical multiobjective flowshop problems involving three simultaneous objectives: (1) minimization of makespan, (2) minimization of mean flow time, and also (3) minimization of the mean tardiness of jobs. Note that no good method—neither analytical nor heuristic—exists in the literature today that finds Pareto-optimal solutions to such problems.

The two multiobjective GA methods, namely NSGA (the Nondominated Sorting GA) proposed by Deb and Srinivas (1995) and described in Chapter 8 and ENGA (the Elitist Nondominated Sorting GA) introduced in Chapter 10, are compared in the following pages.

11.1 NSGA VS ENGA: COMPUTATIONAL EXPERIENCE

In order to evaluate their relative performance, both NSGA and ENGA were coded in the C++ language running on a HP 9000/850 system and then tested on an assortment of flowshop problems. In each GA run, pilot runs guided by the design-of-experiments parameterization framework (Chapter 3) were first conducted to establish the optimum GA parameter values that would produce best convergence. We noted no significant difference between NSGA and ENGA for a given problem in the best values of crossover and mutation probabilities found. Population size was kept unchanged at 100 in all the runs. The value of σ_{share} was kept fixed at 1 to permit

even closely proximate Pareto solutions to show up on the Pareto front.

Two separate comparisons of NSGA and ENGA were made.

The first comparison was exploratory and it aimed at qualitative inference. For it, twenty test problems involving different numbers of machines and jobs were randomly generated. These problems were then each tackled, employing identical starting population of solutions, by NSGA and ENGA respectively (Jayaram, 1998). The progress in finding unique Pareto optimal solutions as the algorithms executed formed the basis for comparison in each case. Subsequently, a 49-job 15-machine flowshop problem was also solved by NSGA and ENGA, both executed for 250 successive GA generations.

In the second comparison, a null hypothesis that there is no difference between the number of Pareto optimal solutions discovered by NSGA and ENGA (i.e., $H_0: \mu_{NSGA} = \mu_{ENGA}$) was statistically tested. This statistical test used the numerical data generated by the earlier completed GA runs.

The processing times for the problems used in these studies were produced randomly from uniform distribution in the interval 5 and 20 while their due dates were generated by dividing the estimated average makespan equally among the jobs. Input data for five typical test problems are shown in Tables 11.2 to 11.6, later in this chapter. The processing time and due date data for the 49-job 15-machine problem appear in Table 4.1.

Figures 11.1 to 11.5 visually display the convergence behavior of NSGA and ENGA presented pairwise for the five flowshop problems (Tables 11.2-11.6). Each random seed marked in these tables triggered the creation of a different initial GA population, however, the initial populations (controlled by the specification of the seed) would provide *identical* starting points for both NSGA and ENGA in the spirit of variance reduction (Law and Kelton, 1991).

To compare their relative performance on a plant scale flowshop problem (one with $49!$ different possible solutions) both NSGA and ENGA were then applied to the 49-job 15-machine problem of Table 4.1. Tables 9.1 and 10.3 present the convergence data for NSGA and ENGA for different starting random seeds, each seed having been

used to provide *identical initial random population* of solutions to NSGA and ENGA. The two lines in Figure 11.6 record the average progress made in finding Pareto optimal solutions by these two algorithms. Figure 11.7 displays the overall relative performance (averaged over various identical initial populations used) of NSGA and ENGA in finding Pareto optimal solutions for the 49-job 15-m/c problem.

The combined results of randomly testing tri-objective flowshop problems support the hypothesis that ENGA would seek out Pareto solutions faster than NSGA and it populates the Pareto optimal front faster than NSGA.

Tables 11.1 and 11.7 contain the summary of typical test results produced by NSGA and ENGA. The data displayed are the count of unique Pareto optimal solutions found, averaged over many random seeds creating the initial populations.

The above comparison hinted that ENGA might actually be more efficient in discovering Pareto solutions. This hypothesis was tested statistically as follows.

11.2 STATISTICAL EVALUATION OF GA RESULTS

To statistically verify the hinting above, the Wilcoxon signed-rank statistical test was applied to the results obtained in Table 11.7. This test is applicable to paired observations (X_{1j}, X_{2j}) , $j = 1, 2, \dots, n$, from two continuous distributions that differ only with respect to their means. It is not necessary here that the distributions be symmetrical. This assures that the distribution of the difference $D_j = X_{1j} - X_{2j}$ is continuous and symmetric (Gibbons, 1996). In the actual test, the null hypothesis $H_0 : \mu_1 = \mu_2$ (which is equivalent to $H_0 : \mu_{\text{Difference}} = 0$) is compared with one-sided alternative hypothesis $H_1 : \mu_1 > \mu_2$ (i.e. $H_1 : \mu_{\text{Difference}} > 0$).

In order to use the signed-rank test, the differences in the unique-Pareto-optimal-solutions-found counts were first ranked in the ascending order of their absolute values. Then the ranks were given the signs of the differences. Ties were assigned average ranks. Now, if W^- is the absolute value of sums of the negative ranks, then, for a

sample size larger than 20, W^- is approximately normally distributed with mean

$$\mu_{W^-} = \frac{n(n+1)}{4}$$

and variance

$$\sigma_{W^-}^2 = \frac{n(n+1)(2n+1)}{24}$$

The critical values C_1 and C_2 for the rejection of $H_0 : \mu_{\text{Difference}} = 0$ are chosen from the standard normal distribution table. If the observed value of the test statistic is $\leq C_1$ or $\geq C_2$, the null hypothesis H_0 is rejected. Table 11.1 displays a summary of typical test results.

In the test involving 21 randomly chosen flowshop problems that used identical initial populations for NSGA and ENGA, in 19 problems ENGA produced numerically larger number of Pareto optimal ("efficient") schedules. Only for two problems NSGA produced more "efficient" solutions. Consequently, one rejects the hypothesis that NSGA and ENGA produce equal number of Pareto optimal solutions when executed for the same number of generations. Therefore, one would regard ENGA to be more efficient in finding Pareto optimal solutions.

The overall quality of solutions produced by ENGA for the test problems evaluated was indistinguishable from those produced by NSGA, i.e. both ENGA and NSGA produced solutions with comparable *degree* of convergence in makespan, mean flow time and mean tardiness when with identical computational effort was employed.

But, as noted, ENGA produced *many more* "efficient" solutions (members of the Pareto optimal front) in the majority of problems, for the same effort (measured as the number of solutions evaluated as the GA executed).

11.3 CHAPTER SUMMARY

Two meta-heuristic methods for producing Pareto optimal schedules for multiobjective flowshops were presented in Chapters 9 and 10. Both these methods use the concept of sharing to induce niche formation so as to encourage Pareto optimal solutions to surface and then remain in the population. ENGA is elitist—it safeguards good solutions found in one generation and saves them to the next.

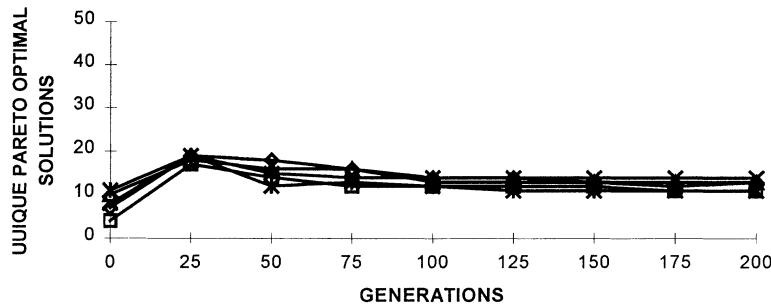
This chapter has presented a statistical comparison of the relative performance of NSGA and ENGA. On the basis this test, ENGA appears to be more efficient in finding the Pareto optimal solutions. Thus, even if the final solutions found by NSGA and ENGA are comparable in quality (both deliver comparable degrees of convergence of the objectives—minimization of makespan, minimization of mean flow time and the minimization of mean tardiness), on the average ENGA finds many more Pareto solutions for the same computational effort and this performance improves with problem size.

TABLE 11.1 WILCOXON SIGNED-RANK TEST: TYPICAL RESULTS

(Null Hypothesis Tested is $\mu_{\text{NSGA}} = \mu_{\text{ENGA}}$)

Problem	μ_w	σ_w	C_1	C_2	w	Result
5M10J-1	976.5	142.63	696.94	1256.06	195	Reject
5M10J-2	105	26.79	52		60.5	Accept
5M10J-3	315	61.05	195.33	434.66	180	Reject
5M20J-1	370.5	68.95	235	505	40.5	Reject
5M20J-3	410	74.40	264.18	555.8	-1	Reject
5M20J-4	370.5	68.95	235.35	505.65	-211	Reject
10M10J-1	1501.5	196.95	1107.6	1895.40	705	Reject
10M10J-2	1620	208.95	1211.35	2028.65	796	Reject
10M10J-3	1463	193.15	1084.42	1841.57	234	Reject
10M10J-4	1540.5	200.77	1146.98	1934.01	30.5	Reject
10M10J-5	1540.5	200.77	1146.98	1934.01	1502	Accept
10M20J-1	410	74.40	264.18	555.82	3	Reject
10M20J-2	390	71.66	249.55	530.45	55.5	Reject
10M20J-3	410	74.40	264.18	555.82	1	Reject
10M20J-4	264	53.48	159.18	368.82	56	Reject
10M20J-5	390	71.66	249.55	530.45	10.5	Reject
15M49J	32490	1975.91	28617.2	36362	274	Reject

NSGA(P5M10J4)



ENGA(P5M10J4)

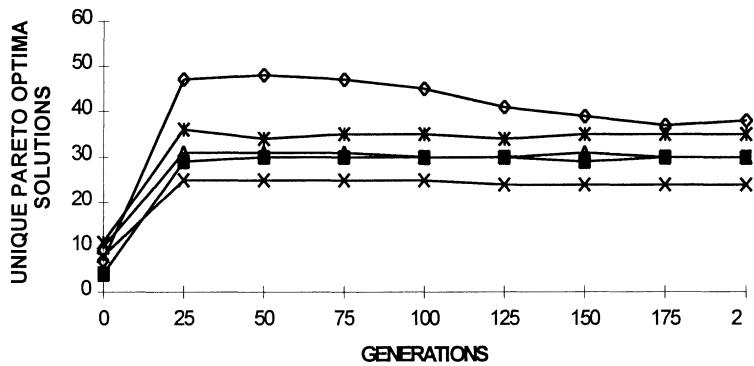
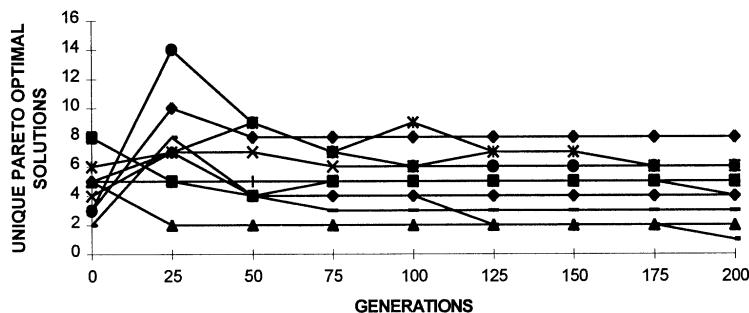


FIGURE 11.1 NSGA vs. ENGA: PROGRESS IN FINDING PARETO OPTIMAL SOLUTIONS FOR PROBLEM P5M10J4

TABLE 11.2 PROCESSING TIMES AND DUE DATES FOR TEST PROBLEM P5M10J4

	m/c #1	2	3	4	5	Due Date
Job 1	15	15	8	6	17	21
2	18	16	11	11	16	42
3	15	19	12	17	10	63
4	16	16	20	16	9	84
5	7	8	5	11	12	105
6	15	20	9	18	13	126
7	7	11	11	9	15	147
8	16	14	13	20	14	168
9	16	11	11	17	8	189
10	6	8	10	14	19	210

NSGA (P10M10J3)



ENGA (P10M10J3)

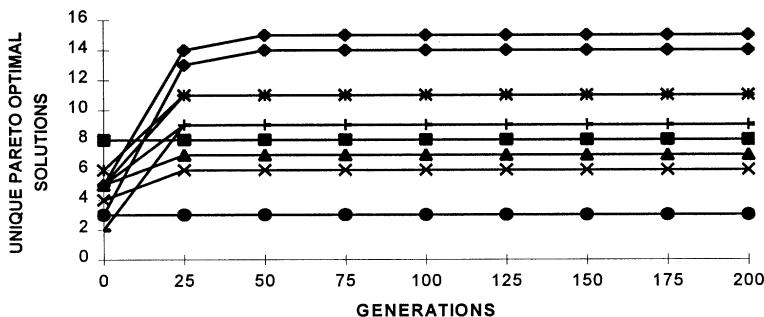
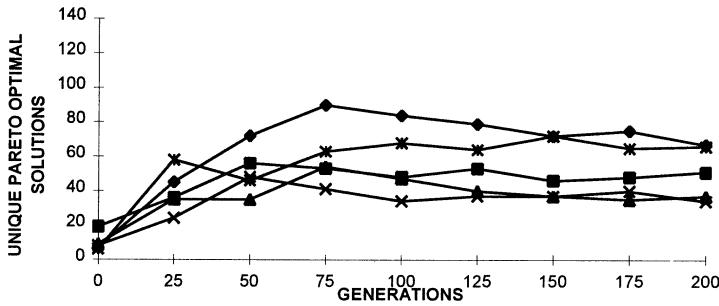


FIGURE 11.2 PROGRESS IN FINDING PARETO SOLUTIONS FOR P10M10J3

TABLE 11.3 PROCESSING TIMES AND DUE DATES FOR PROBLEM P10M10J3

Job	m/c										Due Date
	#1	2	3	4	5	6	7	8	9	10	
1	10	16	13	18	15	16	9	14	7	13	26
2	9	8	6	16	10	8	5	11	5	18	52
3	16	8	9	7	19	5	19	12	13	10	78
4	11	7	14	20	8	12	18	13	14	14	104
5	5	8	9	18	18	20	19	13	6	20	130
6	7	11	16	11	18	16	9	16	15	10	156
7	9	15	15	7	16	16	12	8	19	11	182
8	16	16	6	6	5	6	9	18	13	19	208
9	20	13	15	5	7	9	17	5	8	15	234
10	11	20	19	12	19	16	10	12	11	15	260

NSGA(P5M20J3)



ENGA(P5M20J3)

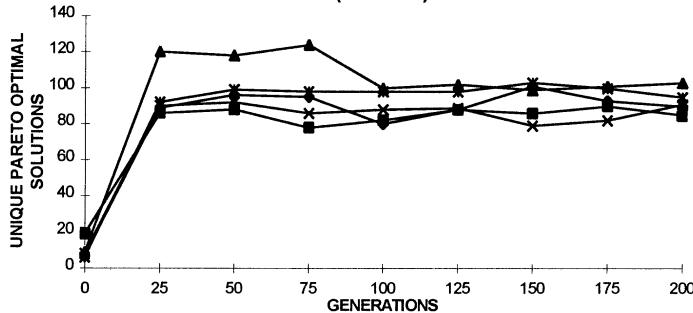
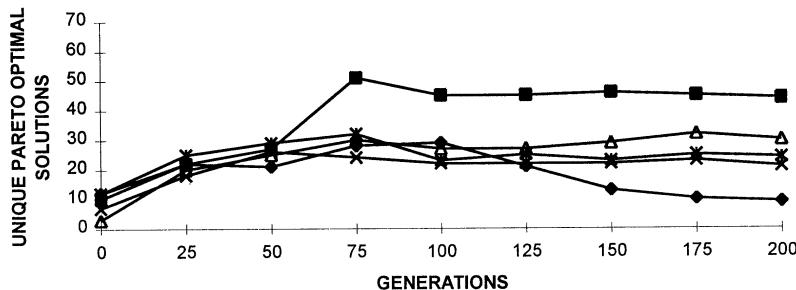


FIGURE 11.3 PROGRESS IN FINDING PARETO SOLUTIONS FOR P5M20J3

TABLE 11.4 PROCESSING TIMES AND DUE DATES FOR P5M20J3

	m/c # 1	2	3	4	5	Due Date
Job 1	18	16	8	8	9	19
2	9	10	18	5	11	38
3	18	15	9	7	8	57
4	7	16	12	9	8	76
5	19	18	13	17	7	95
6	14	17	8	10	12	114
7	6	17	20	9	9	133
8	18	17	13	13	16	152
9	16	20	20	5	20	171
10	12	19	10	5	8	190
11	7	6	20	12	5	209
12	11	16	16	7	13	228
13	19	15	19	14	17	247
14	8	14	10	11	5	266
15	18	20	8	16	19	285
16	14	13	5	12	13	304
17	6	20	16	6	20	323
18	11	18	13	5	9	342
19	12	11	14	5	6	361

NSGA(P5M20J5)



ENGA(P5M20J5)

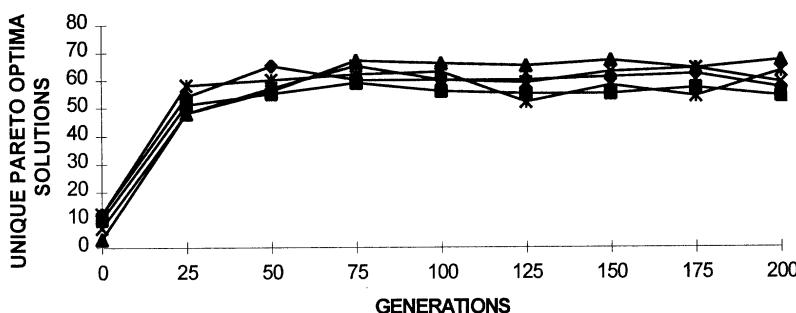


FIGURE 11.4 PROGRESS IN FINDING PARETO SOLUTIONS FOR 5M20J5

TABLE 11.5 PROCESSING TIMES AND DUE DATES FOR PROBLEM P5M20J5

	m/c #1	2	3	4	5	Due Date
Job 1	9	7	11	5	9	18
2	12	11	19	15	15	36
3	20	7	9	13	5	54
4	13	17	13	11	19	72
5	11	8	16	7	12	90
6	14	14	10	14	11	108
7	15	6	11	13	17	126
8	12	10	10	19	14	144
9	16	5	14	6	16	162
10	8	15	20	15	9	180
11	13	17	19	7	7	198
12	8	9	15	8	11	216
13	16	11	19	11	20	234
14	19	11	13	14	17	252
15	8	11	11	17	13	270
16	15	9	17	7	14	288
17	7	17	16	19	18	306
18	7	15	16	5	10	324
19	20	20	7	9	10	342
20	10	12	16	5	17	360

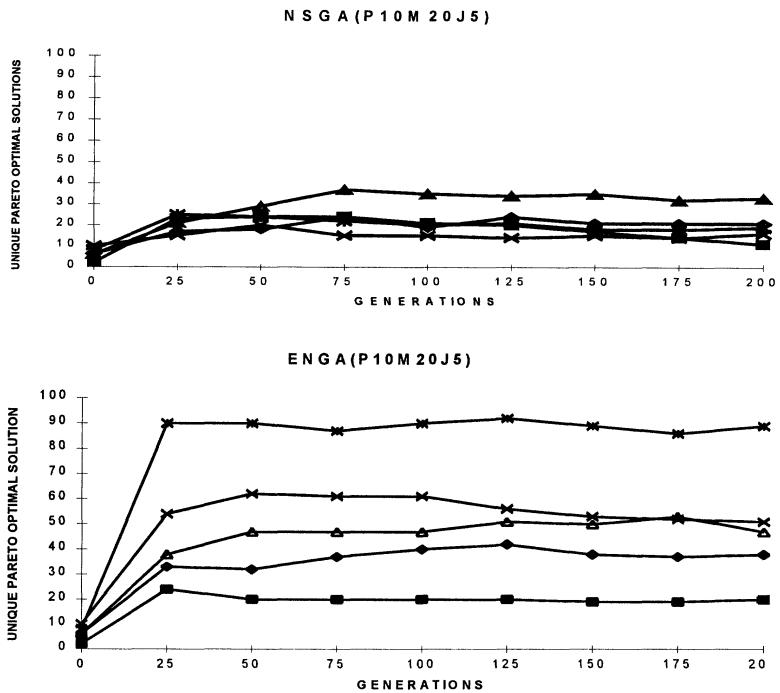


FIGURE 11.5 PROGRESS IN FINDING PARETO OPTIMAL SOLUTIONS FOR PROBLEM P10M20J5

TABLE 11.6 PROCESSING TIMES AND DUE DATES FOR PROBLEM P10M20J5

	m/c #1	2	3	4	5	6	7	8	9	10	Due Date
Job 1	9	7	11	5	9	17	20	10	18	6	22
2	12	11	19	15	15	14	16	19	14	17	44
3	20	7	9	13	5	9	17	7	6	20	66
4	13	17	13	11	19	10	12	18	6	5	88
5	11	8	16	7	12	16	8	17	10	5	110
6	14	14	10	14	11	5	15	8	14	7	132
7	15	6	11	13	17	6	12	15	6	7	154
8	12	10	10	19	14	19	5	12	15	16	176
9	16	5	14	6	16	7	15	6	9	14	198
10	8	15	20	15	9	5	12	13	14	18	220
11	13	17	19	7	7	9	6	17	6	19	242
12	8	9	15	8	11	10	9	20	6	12	264
13	16	11	19	11	20	9	11	14	5	8	286
14	19	11	13	14	17	13	11	7	11	10	308
15	8	11	11	17	13	8	8	5	7	18	330
16	15	9	17	7	14	12	11	20	16	5	352
17	7	17	16	19	18	7	12	18	9	8	374
18	7	15	16	5	10	7	16	12	14	7	396
19	20	20	7	9	10	10	8	16	11	11	418
20	10	12	16	5	17	12	16	16	17	16	440

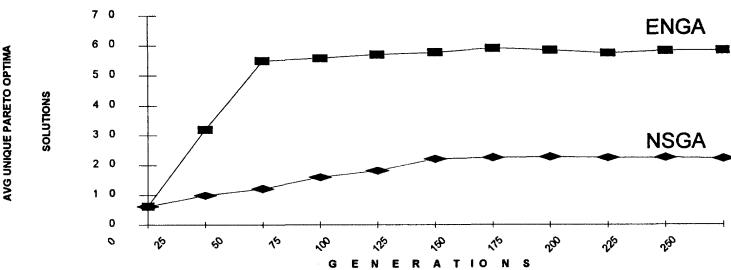


FIGURE 11.6 PROGRESS IN FINDING PARETO OPTIMAL SOLUTIONS FOR THE 49-JOB 15-MACHINE FLOWSHOP
(Plots display the average result of using 37 different random seeds.)

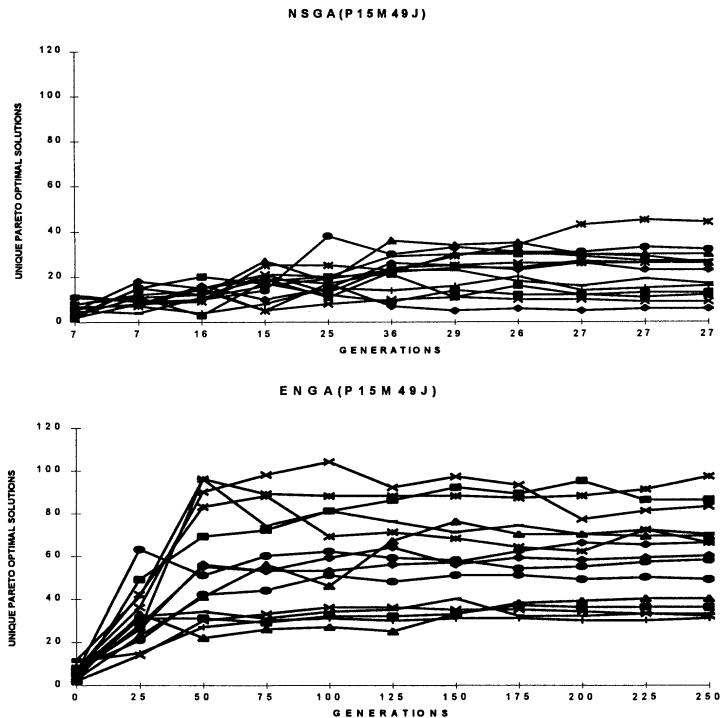


FIGURE 11.7 NSGA vs. ENGA : PROGRESS IN FINDING PARETO OPTIMAL SOLUTIONS FOR THE 49-JOB 15-MACHINE PROBLEM

TABLE 11.7 SAMPLES OF THE AVERAGE NUMBER OF PARETO SOLUTIONS FOUND BY NSGA AND ENGA FOR 21 FLOWSHOP PROBLEMS TESTED

Problem ID	Algorithm	Gen 0	25	50	75	100	125	150	175	200
5M10J-1	NSGA	5.25	15.75	13.37	12.5	12	11.87	11.5	11.12	10.62
5M10J-1	ENGA	5.25	17.75	17.75	18.12	18.12	18.12	18.12	18	18.12
5M10J-2	NSGA	3.8	6.8	5.6	4.8	4.6	4.4	4.4	4.2	4.2
5M10J-2	ENGA	3.8	5.2	5.2	5.2	5.2	5.2	5.2	5.2	5.2
5M10J-3	NSGA	5.8	27	27.2	26.4	26.4	26	26.4	24	25.6
5M10J-3	ENGA	5.8	27.8	27.4	27.6	28.2	28	27.6	28.4	28.2
5M10J-4	NSGA	8	18.4	15	14.2	13	12.8	12.6	12.2	12.4
5m10J-4	ENGA	8	33.6	33.6	33.6	33	31.8	31.6	31.2	31.4

12

MULTIOBJECTIVE JOB SHOP SCHEDULING

This chapter demonstrates the use of nondominated sorting GAs to schedule *multiobjective job shops* (JSP). The illustration uses a JSP involving three separate management objectives: (1) minimization of makespan, (2) minimization of mean flow time, and also (3) minimization of the mean tardiness of jobs. As is the case with the flowshop, no method—neither analytical nor heuristic—exists in the literature today for developing Pareto-optimal solutions to this scheduling problem. The two nondominated sorting GAs used here are, as done in the previous chapter, NSGA and ENGA respectively. As is the case with the flowshop, the incorporation of elitism makes the discovery of Pareto optimal schedules more efficient.

12.1 MULTIOBJECTIVE JSP IMPLEMENTATION

In this implementation the operation-based JSP chromosome representation originally used by Gen, Tsujimura and Kubota (1994) described in Section 5.3.1 (Figure 5.3) is employed. As stated there, this representation encodes a schedule as a sequence of operations in which each gene stands for one operation. All the operations for a job are named with the same symbol (= job #) and they are interpreted as operations required on that job in accordance with their order of occurrence along the length of the chromosome, left to right. For example, for a three-job three-machine JSP, a chromosome would be as shown in Figure 12.1.

The material below, which includes the JSP chromosome used in this chapter and the mutation and crossover operations specific to it, is reproduced from Chapter 5 because of its significance in coding the JSP problem for NSGA.

12.1.1 Chromosome Representation

Gen, Tsujimura, and Kubota (1994) devised a clever *operation-based* representation for solving the JSP. This representation encodes a JSP schedule as a sequence of operations in which each gene stands for one *operation*. In their scheme, Gen et al. name all operations of a job with identical symbols and then interpret them according to their *order* of occurrence in the sequence (left to right) in the chromosome. Recall that in a job shop the sequence of operations on a job are technologically determined and it is not possible to switch their order.

For an n -job m -machine JSP, the chromosome would contain $n \times m$ genes, each job appearing in the chromosome exactly m times. The repeating occurrence of a job # (as an *allele* in a gene) does not indicate a concrete operation of a job but rather it refers to an operation that is *context (technology)-dependent*.

Thus, the operation-based representation encodes a schedule as a sequence of operations on the different jobs, and each gene here stands for one operation. All operations for a job are represented by the same symbol (e.g. "3" for job #3 or J_3) and they are interpreted according to the *order* of their occurrence in the sequence *in which they appear* in the chromosome. A 3-job 3-machine solution representation is shown in Figure 12.1 in which the number "1" stands for an operation for job J_1 , "2" stands for an operation for job J_2 , and "3" stands for an operation for job J_3 .

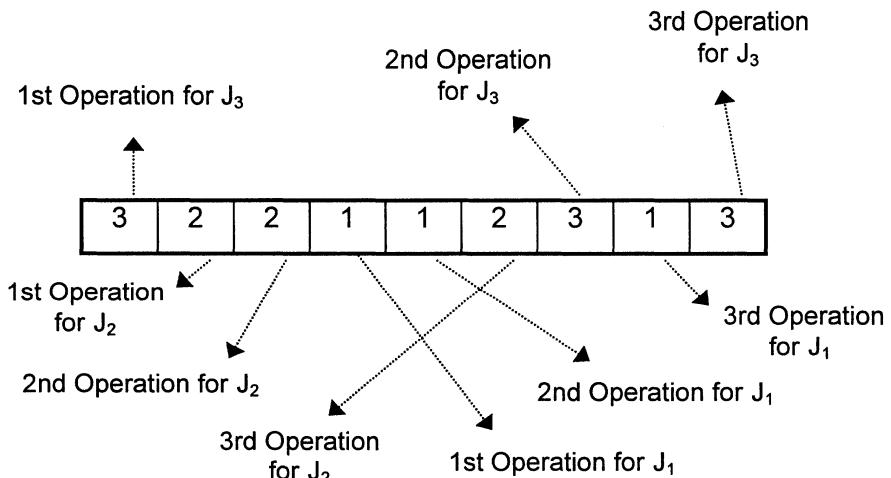


FIGURE 12.1 OPERATION-BASED REPRESENTATION OF A JSP

You may observe that any *permutation* of the genes shown as segments in the chromosome above will always yield a feasible schedule without "repair." This is the key advantage of an operation-based JSP representation. Parent **p1** in Figure 12.2 contains a JSP schedule for a 4-machine 4-job job shop. **p1** contains exactly four "1" alleles (gene values), indicating the *four machine operations* required for job 1. Similarly, it contains four "2"s, four "3"s and four "4"s in it, thus it is a legal schedule.

Figures 12.2 through 12.5 show the steps of crossing such chromosomes and then legalizing the offspring, accomplished as described below.

12.1.2 Crossover

The method uses a *partial schedule exchange* crossover operator which considers the *partial schedules* (e.g. 4 1 2 4 in **p1**) in Figure 12.2 to be the natural building blocks in offspring in much the same manner as Holland (1975) has described his building blocks. A partial schedule is identified with the *same* job in the *first* and *last* positions of the partial schedule. The steps for the crossover operation are described below (refer to Figures 12.2, 12.3, 12.4 and 12.5).

In the illustration given for a 4-machine 4-job problem, we begin Step 1 with two parent chromosomes **p1** and **p2** (shown in Figure 12.2).

- Step 1. Pick one partial schedule (a substring of random length) in the first parent—randomly. The first position in this partial schedule is the 6th position. At this position job 4 is located (Figure 12.2).
- Step 2. Find the next-nearest job 4 in the same parent **p1**, which is in position 9. Thus the *partial schedule 1* formed is (4 1 2 4).
- Step 3. Locate the second partial schedule in parent **p2** as follows. It must be the first substring to begin and end with job 4 (as is true for first partial schedule). Thus partial schedule 2 is formed with the genes between the first operation of job 4 and the second operation of job 4 in parent **p2**. This partial schedule is 4 1 3 1 1 3 4 .

Step 4. Exchange the partial schedules 4 1 2 4 and 4 1 3 1 1 3 4 to produce progeny **o1** and **o2**. Figure 12.3 shows the result of this crossover exchange.

Usually, the partial schedules being exchanged would contain a different number of genes, so the offspring generated after exchanging might be illegal. An illegal offspring may not include or may have excess of all the operations required for each job. The "missed" and "exceeded" genes in an offspring indicate an illegal schedule as shown in Figure 12.4. The next step legalizes (repairs) each offspring by deleting exceeded genes and adding missed genes caused by exchange of partial schedules between **p1** and **p2**.

In the example, by crossover, offspring **o1** gained extra genes 3, 1, 1 and 3 while it lost gene 2. Therefore, the extra genes (3, 1, 1 and 3) are to be deleted, and the missing gene(s) (here 2) are to be inserted in **o1** (in the position immediately after the newly acquired partial schedule)—to make **o1** legal. Repair is then repeated for **o2** to render offspring **o2** also legal.

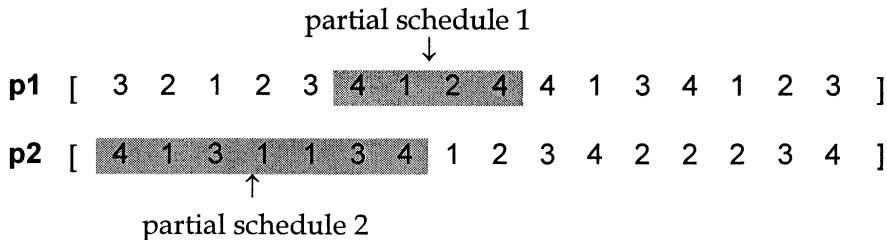


FIGURE 12.2 STEP 1: SELECT PARTIAL SCHEDULES

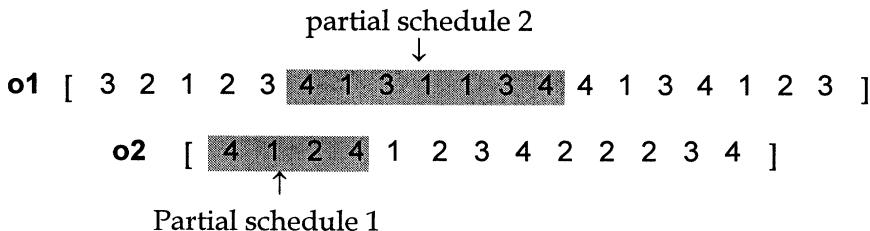


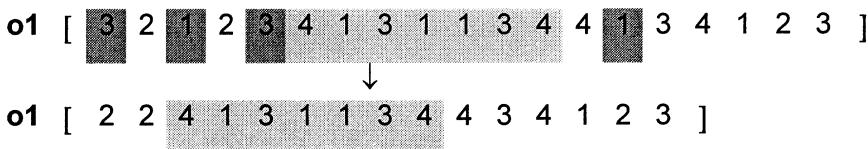
FIGURE 12.3 STEP 2: EXCHANGE PARTIAL SCHEDULES

Partial Schedule 1 4 1 2 4
 replaced in **p1** by 4 1 3 1 1 3 4
 \Rightarrow gene "2" missing in offspring **o1**, exceeded genes are "3," "1," "1" and "3."

Partial Schedule 2 4 1 3 1 1 4
 replaced in **p2** by 4 1 2 4
 \Rightarrow genes "3," "1," "1," "3," missing in offspring **o2**; exceeded gene is "2."

FIGURE 12.4 STEP 3: IDENTIFY MISSED/EXCEEDED GENES

Randomly Delete the exceeded genes "3," "1," "1" and "3" without disturbing the inserted partial schedule



Insert the missed gene "2" after the inserted partial schedule

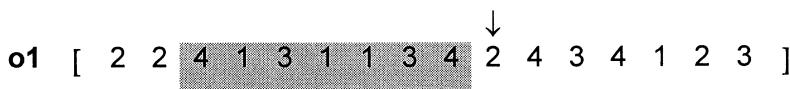


FIGURE 12.5 CROSSOVER STEP 4: LEGALIZE OFFSPRING o1

12.1.3 Mutation

The mutation operator used in this approach is the *job-pair exchange* mutation, that is, two non-identical jobs are picked randomly and then they exchange their positions as shown in Figure 5.8. The results are legal, hence no repair is necessary here.

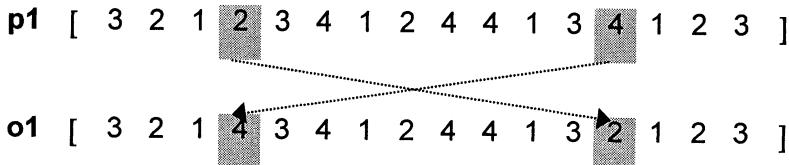


FIGURE 12.6 THE JOB-PAIR EXCHANGE MUTATION

12.2 NSGA VS ENGA: COMPUTATIONAL EXPERIENCE

In order to evaluate their relative performance, both NSGA and ENGA were coded in the C++ language executing on an HP 9000/850 system and tested on an assortment of job shop problems. Four different problem sizes were tested. The processing times and due dates for typical problems solved are shown in Tables 12.1 through 12.4. The processing times here were picked randomly from a uniform distribution in the interval 5 and 20. The due dates were fixed (arbitrarily) by dividing the estimated average makespan equally among the jobs being scheduled.

Figures 12.7 to 12.10 graphically display the convergence behaviour of NSGA and ENGA, averaged for ten different starting random seeds, for the four above JSP problems. In each case, NSGA and ENGA were started with identical randomly picked initial solutions. It is evident that ENGA was generally able to seek out the Pareto optimal solutions faster and it also was able to populate the Pareto front faster. Subsequently, test data when subjected to the Wilcoxon signed-rank test supported the rejection of the hypothesis: $\text{Performance}_{\text{ENGA}} = \text{Performance}_{\text{NSGA}}$.

12.3 CHAPTER SUMMARY

This chapter has presented two methods for finding Pareto solutions to multiobjective JSPs. ENGA appears to be more efficient. ENGA may be further improved by changing its solution coding method (see Dorndorf and Pesch, 1995) and hybridizing it, for instance, by using in it the shifting bottleneck algorithm (Adams, Balas and Zawak, 1988).

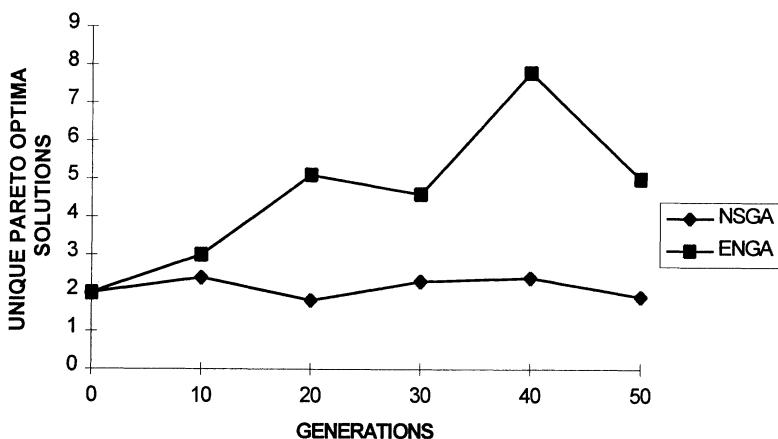


FIGURE 12.7 NSGA vs. ENGA: RELATIVE PROGRESS IN FINDING PARETO OPTIMAL SOLUTIONS FOR THE JSP SHOWN IN TABLE 12.1

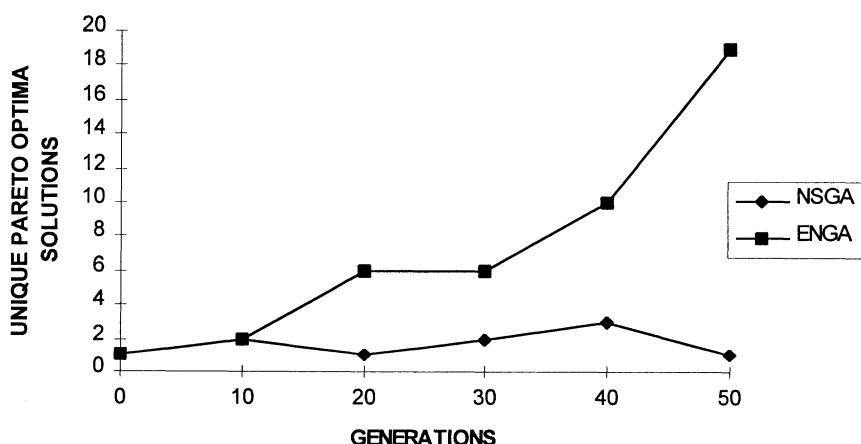


FIGURE 12.8 NSGA vs. ENGA: RELATIVE PROGRESS IN FINDING PARETO OPTIMAL SOLUTIONS FOR THE JSP SHOWN IN TABLE 12.2

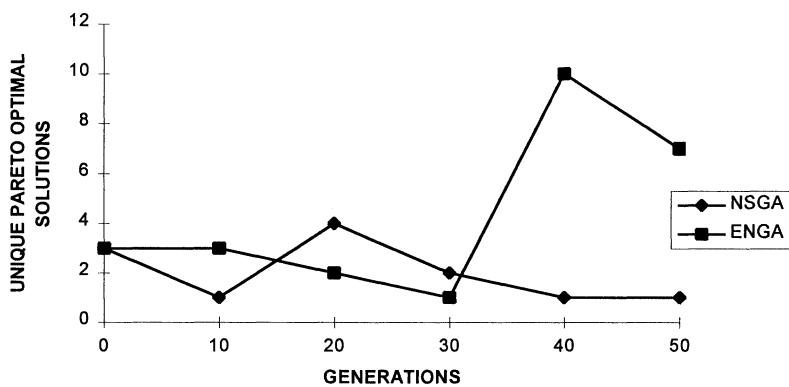


FIGURE 12.9 NSGA vs. ENGA: RELATIVE PROGRESS IN FINDING PARETO OPTIMAL SOLUTIONS FOR THE JSP SHOWN IN TABLE 12.3

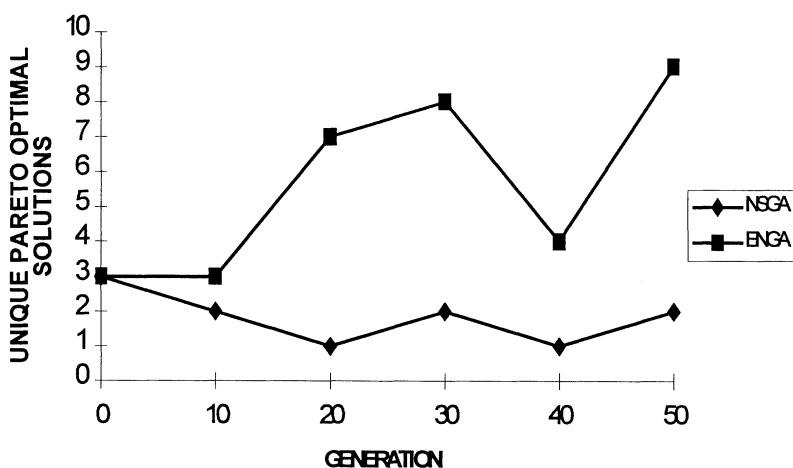
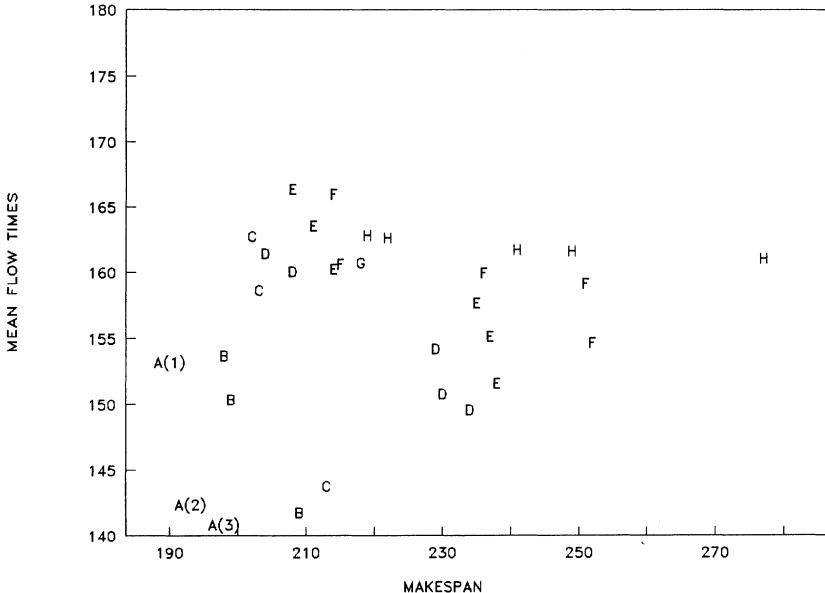


FIGURE 12.10 NSGA vs. ENGA: RELATIVE PROGRESS IN FINDING PARETO OPTIMAL SOLUTIONS FOR THE JSP SHOWN IN TABLE 12.4



Pareto Optimal solutions to the 3-objective 10-job, 5 m/c JSP of Table 12.1

Schedule A(1): Makespan 190, Mean Flow Time 153.2,
Mean Tardiness 29.8

Job #	7	5	2	3	8	4	5	1	5	9	3	5	10	10	7	10	10	9	8	8	2	6	1	8	4
Operation #	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
Job #	1	9	3	6	4	2	2	7	6	1	4	10	9	8	4	1	5	7	9	6	2	3	7	6	3
Operation #	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50

Schedule A(2): Makespan 198, Mean Flow Time 140.9,
Mean Tardiness 20.1

Job #	1	3	10	2	8	9	2	5	9	8	6	8	7	3	10	1	7	1	5	9	4	4	1	10	3
Operation #	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
Job #	1	9	7	9	7	5	5	3	4	2	6	8	10	10	7	5	4	3	8	6	2	6	4	2	6
Operation #	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50

Schedule A(3): Makespan 193, Mean Flow Time 142.4,
Mean Tardiness 22.8

Job #	4	8	2	2	10	3	5	9	10	4	9	1	6	10	5	10	3	4	4	10	2	3	2	9	7
Operation #	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
Job #	4	3	2	1	9	9	8	1	6	3	8	5	6	5	5	7	7	6	1	7	6	8	7	8	1
Operation #	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50

TABLE 12.1 DUE DATES AND PROCESSING TIMES FOR JSP PROBLEM #1

	(m/c, t)	Due Date				
Job 1	1, 13	5, 16	4, 19	2, 7	3, 14	37
2	4, 19	5, 7	2, 13	1, 17	3, 19	74
3	3, 19	2, 18	5, 16	4, 18	1, 19	111
4	1, 14	4, 15	5, 10	2, 13	3, 17	148
5	1, 8	2, 8	5, 19	4, 7	3, 9	185
6	3, 16	2, 15	5, 20	4, 18	1, 10	222
7	2, 14	4, 17	3, 18	1, 5	5, 20	259
8	1, 8	2, 6	4, 9	5, 20	3, 7	296
9	5, 16	4, 13	3, 9	2, 16	1, 12	333
10	2, 12	1, 19	3, 9	5, 6	4, 7	370

TABLE 12.2 DUE DATES AND PROCESSING TIMES FOR JSP PROBLEM #2

	(m/c, t)	Date Date							
Job 1	2, 13	9, 16	1, 17	3, 7	6, 14	7, 19	4, 13	8, 9	10, 12
2	7, 12	5, 19	10, 5	9, 6	8, 15	4, 15	6, , 12	2, 8	1, 6
3	8, 8	7, 8	6, 10	9, 20	4, 12	1, 10	3, 15	5, 15	10, 5
4	3, 8	10, 20	4, 14	2, 19	8, 17	6, 18	5, 9	9, 20	7, 6
5	5, 20	1, 13	3, 12	8, 7	9, 9	7, 10	2, 8	10, 16	4, 6
6	8, 6	3, 14	10, 19	4, 14	9, 20	2, 12	1, 16	6, 17	7, 16
7	2, 16	3, 6	8, 7	10, 17	7, 13	1, 17	5, 17	6, 17	4, 14
8	4, 6	6, 13	3, 14	1, 9	10, 13	5, 13	7, 19	2, 15	8, 18
9	7, 8	2, 13	3, 13	9, 7	10, 9	5, 10	6, 16	1, 11	4, 16
10	3, 13	4, 18	1, 19	5, 6	10, 5	9, 19	2, 19	6, 6	8, 9
									7, 18
									340

TABLE 12.3 DUE DATES AND PROCESSING TIMES FOR JSP PROBLEM #3

	(m/c, t)	Due Date				
Job 1	1, 13	5, 16	4, 19	2, 7	3, 14	18
2	4, 19	5, 7	2, 13	1, 17	3, 19	36
3	3, 19	2, 18	5, 16	4, 18	1, 19	54
4	1, 14	4, 15	5, 10	2, 13	3, 17	72
5	1, 8	2, 8	5, 19	4, 7	3, 9	90
6	3, 16	2, 15	5, 20	4, 18	1, 10	108
7	2, 14	4, 17	3, 18	1, 5	5, 20	126
8	1, 8	2, 6	4, 9	5, 20	3, 7	144
9	5, 16	4, 13	3, 9	2, 16	1, 12	162
10	2, 12	1, 19	3, 9	5, 6	4, 7	180
11	1, 19	2, 16	4, 10	5, 16	3, 6	198
12	4, 7	5, 13	1, 8	2, 14	3, 7	216
13	5, 6	4, 14	1, 16	2, 19	3, 20	234
14	2, 10	4, 16	1, 19	5, 13	3, 18	252
15	5, 19	1, 8	3, 8	2, 14	4, 9	270
16	1, 16	5, 17	4, 13	3, 17	2, 10	288
17	4, 15	1, 15	3, 8	2, 10	5, 20	306
18	5, 15	3, 13	2, 15	4, 9	1, 17	324
19	4, 8	3, 8	1, 15	5, 15	2, 6	342
20	2, 13	3, 9	1, 9	5, 13	4, 19	360

TABLE 12.4 DUE DATES AND PROCESSING TIMES FOR JSP PROBLEM #4

	(m/c, t)	Due Date												
Job 1	2, 13	9, 16	1, 17	3, 7	6, 14	7, 19	4, 13	8, 9	10, 12	5, 20				26
2	7, 12	5, 19	10, 5	9, 6	8, 15	4, 15	6, , 12	2, 8	1, 6	3, 12				52
3	8, 8	7, 8	6, 10	9, 20	4, 12	1, 10	3, 15	5, 15	10, 5	2, 17				78
4	3, 8	10, 20	4, 14	2, 19	8, 17	6, 18	5, 9	9, 20	7, 6	1, 8				104
5	5, 20	1, 13	3, 12	8, 7	9, 9	7, 10	2, 8	10, 16	4, 6	6, 16				130
6	8, 6	3, 14	10, 19	4, 14	9, 20	2, 12	1, 16	6, 17	7, 16	5, 6				156
7	2, 16	3, 6	8, 7	10, 17	7, 13	1, 17	5, 17	6, 17	4, 14	9, 15				182
8	4, 6	6, 13	3, 14	1, 9	10, 13	5, 13	7, 19	2, 15	8, 18	9, 16				208
9	7, 8	2, 13	3, 13	9, 7	10, 9	5, 10	6, 16	1, 11	4, 16	8, 18				234
10	3, 13	4, 18	1, 19	5, 6	10, 5	9, 19	2, 19	6, 6	8, 9	7, 18				260
11	1, 18	3, 15	8, 12	6, 8	7, 14	10, 11	2, 10	9, 11	5, 12	4, 14				286
12	6, 20	5, 13	3, 19	10, 15	7, 15	1, 16	9, 13	8, 8	4, 6	2, 12				312
13	3, 5	1, 20	2, 20	7, 10	4, 11	10, 19	6, 11	5, 14	8, 10	9, 9				338
14	7, 5	3, 13	10, 16	1, 20	4, 19	9, 16	6, 20	8, 8	5, 9	2, 18				364
15	2, 5	8, 6	6, 12	4, 15	1, 12	7, 8	3, 13	5, 10	10, 6	9, 10				390
16	4, 17	5, 12	10, 18	3, 20	7, 8	2, 9	8, 6	1, 19	6, 11	9, 18				416
17	9, 13	5, 5	2, 9	7, 12	3, 5	8, 7	1, 18	6, 9	4, 13	10, 19				442
18	5, 13	4, 12	9, 9	2, 19	10, 6	7, 20	6, 12	8, 5	3, 14	1, 10				468
19	8, 10	9, 11	10, 14	4, 10	3, 18	6, 11	2, 12	7, 19	1, 20	5, 12				494
20	8, 8	6, 18	9, 11	4, 18	7, 14	5, 7	10, 17	1, 5	3, 5	2, 18				520

MULTIOBJECTIVE OPEN SHOP SCHEDULING

This chapter presents the results of scheduling the multiobjective *open shop* (OSSP). The study is conducted using two conflicting OSSP objectives: (1) minimization of makespan, and (2) minimization of the mean flow time of jobs.

The open shop, it may be recalled, is a ubiquitous problem, occurring in many job shops in which the tasks for a particular job may be carried out in (almost) any order, such as automotive repairs (tasks) for cars (jobs), or upgrades/repairs (tasks) for PCs (jobs). We note again that no method, neither analytical nor heuristic, exists in the literature today for providing Pareto-optimal solutions to open shop scheduling. The two nondominated sorting GA-based approaches (NSGA and ENGA) to finding Pareto-optimal schedules are compared in the following pages in the context of the OSSP.

13.1 AN OVERVIEW OF THE OPEN SHOP

A single-stage production system requires one operation for each job. A multi-stage system involves jobs that require operations on different machines. Three main types of multi-stage systems are commonly encountered. In all such systems m machines are present, each with a different function. In a flowshop with m stages, each job is processed on machines 1, 2, 3,..., m in that order. In a job shop, each job has a prescribed routing through the machines, determined by the technological requirements for processing each job, and this routing may differ from job to job. In an open shop, each job is processed once on each machine, but the machine routing (that specifies the sequence of machines through which a job must pass) can differ between jobs and determining such optimal routings for all the jobs to be processed forms a part of the scheduling decision process. Owing to this

flexibility, the OSSP has a considerably larger search space than the JSP.

Thus, the open shop scheduling problem is similar to the job shop scheduling problem (JSP), with the exception that there is no *a priori* ordering in the open shop on the tasks within any job. Therefore, the open shop is often closest to the real world of scheduling. Even timetabling problems may often be formulated as open shop problems.

TABLE 13.1 PROCESSING TIMES FOR A TYPICAL 3-m/c 3-JOB OPEN SHOP

	(m, t)	(m, t)	(m, t)
Job 1	1, 13	3, 16	2, 19
Job 2	3, 19	1, 7	2, 13
Job 3	1, 19	2, 18	3, 16

Table 13.1 shows the requirements for a typical 3-machine 3-job OSSP in which symbols m and t stand for machine number and processing time respectively. In the table, operation 1 of job 1 must be processed on machine 1 for 13 units of time, operation 2 of job 1 must be processed on machine 3 for 16 units of processing time, and so on, however, with no restrictions on the *order* in which the different operations for any job are to be processed. The optimal solution, nevertheless, must specify, in addition to a sequence for processing the jobs for each machine, the machine routing for each job. In the multiobjective open shop, the object may be to generate a schedule that simultaneously minimizes the makespan and the mean flow time of the jobs—two objectives that may in general be in conflict.

13.2 MULTIOBJECTIVE GA IMPLEMENTATION

Since the solution for the open shop must specify a job sequence for each machine and also a machine routing for each job, the *natural* solution representation—a set of sequences, one for each machine—that works well for flow and job shops, cannot work for the open shop. A *priority representation* counters some of these drawbacks. For an open shop, if we specify for each machine a priority order for

the jobs, and each job a sequence of machines that defines its route, the solution representation is complete.

In the present GA implementation the operation-based chromosome representation discussed in Section 12.1 was used. This representation encodes a schedule as a sequence of operations in which each "gene" stands for one operation. Natural numbers are used to name each operation to be performed in the open shop, regardless of which job is involved.

For example, for the three-job three-machine OSSP problem shown in Table 13.1, a natural number may be used to name each of the nine different operations required to complete all the jobs. Thus 1 would represent operation 1 of job1 done on machine 1 with processing time of 13 units, 2 would represent operation 2 of job1 done on machine 3 with processing time of 16 units, and so on. Thus a GA chromosome may represent a solution sequence on natural numbers as shown below

4	7	2	6	1	9	3	8	5
---	---	---	---	---	---	---	---	---

where each symbol (4, 7, 2, etc.) stands for a unique and distinct operation for a particular job on a specified machine.

The optimal solution in this problem would be a permutation of the natural numbers 1 to 9. (Note that the interpretation of the OSSP solution sequence is entirely different from that of a similar-looking flowshop chromosome! In the open shop the solution specifies the priority in which the jobs are to be loaded and processed on the different machines, many operations starting at once on the different machines.)

The reproduction operator processed the chromosomes using nondomination as the basis. The partially mapped crossover (Uckun, Bagchi, Kawamura and Miyabe, 1993) and job-pair exchange mutation operator described in Chapter 12 were the other genetic operators employed in this implementation.

13.3 NSGA VS ENGA: SOME COMPUTATIONAL RESULTS

In order to evaluate their relative performance both NSGA and ENGA were coded in C++ running on HP 9000/850 and tested on an assortment of randomly generated open shop problems. The processing times for these problems were produced from uniform distribution in the interval 5 and 20. Four such typical problems are shown in Tables 13.2 to 13.5.

Figure 13.1 shows a comparison of the number of unique Pareto optimal solutions produced by NSGA and ENGA, averaged over ten different initial populations. It also shows the progress made in finding the Pareto solutions as GA generations progressed, up to 50 generations. Figures 13.2-13.4 display the data for OSSP problems characterised by Tables 13.3, 13.4 and 13.5 respectively.

In each of these situations ENGA visually indicated superior performance. Statistical tests of the data generated support the hypothesis that ENGA seeks out more unique Pareto optimal solutions than does NSGA for identical computational effort, and ENGA produces a pre-specified number of Pareto solutions faster.

Based on these results we conclude that ENGA is able to seek out the Pareto optimal OSSP solutions faster and it is also able to populate the Pareto front faster than does NSGA.

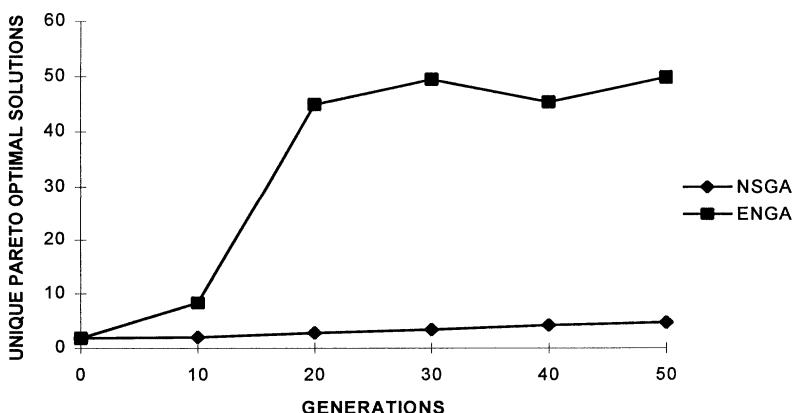


FIGURE 13.1 NSGA vs. ENGA: PROGRESS IN FINDING PARETO OPTIMAL SOLUTIONS FOR THE OSSP IN TABLE 13.2

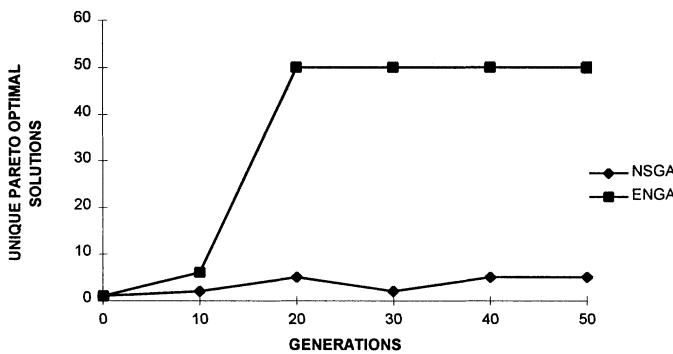


FIGURE 13.2 NSGA vs. ENGA: PROGRESS IN FINDING PARETO OPTIMAL SOLUTIONS FOR THE OSSP IN TABLE 13.3

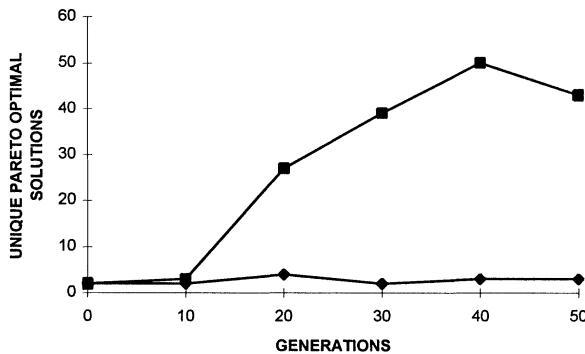


FIGURE 13.3 NSGA vs. ENGA: PROGRESS IN FINDING PARETO OPTIMAL SOLUTIONS FOR THE OSSP IN TABLE 13.4

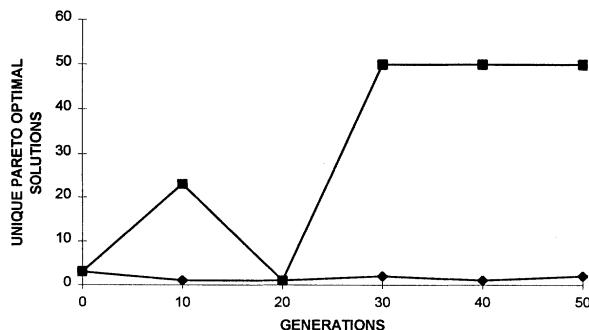


FIGURE 13.4 NSGA vs. ENGA: PROGRESS IN FINDING PARETO OPTIMAL SOLUTIONS FOR THE OSSP IN TABLE 13.5

TABLE 13.2 PROCESSING TIMES FOR A 10-JOB 5-m/c OPEN SHOP

	(m/c, t)				
Job 1	1, 13	5, 16	4, 19	2, 7	3, 14
2	4, 19	5, 7	2, 13	1, 17	3, 19
3	3, 19	2, 18	5, 16	4, 18	1, 19
4	1, 14	4, 15	5, 10	2, 13	3, 17
5	1, 8	2, 8	5, 19	4, 7	3, 9
6	3, 16	2, 15	5, 20	4, 18	1, 10
7	2, 14	4, 17	3, 18	1, 5	5, 20
8	1, 8	2, 6	4, 9	5, 20	3, 7
9	5, 16	4, 13	3, 9	2, 16	1, 12
10	2, 12	1, 19	3, 9	5, 6	4, 7

TABLE 13.3 PROCESSING TIMES FOR A 20-JOB 5-m/c OPEN SHOP

	(m/c, t)				
Job 1	1, 13	5, 16	4, 19	2, 7	3, 14
2	4, 19	5, 7	2, 13	1, 17	3, 19
3	3, 19	2, 18	5, 16	4, 18	1, 19
4	1, 14	4, 15	5, 10	2, 13	3, 17
5	1, 8	2, 8	5, 19	4, 7	3, 9
6	3, 16	2, 15	5, 20	4, 18	1, 10
7	2, 14	4, 17	3, 18	1, 5	5, 20
8	1, 8	2, 6	4, 9	5, 20	3, 7
9	5, 16	4, 13	3, 9	2, 16	1, 12
10	2, 12	1, 19	3, 9	5, 6	4, 7
11	1, 19	2, 16	4, 10	5, 16	3, 6
12	4, 7	5, 13	1, 8	2, 14	3, 7
13	5, 6	4, 14	1, 16	2, 19	3, 20
14	2, 10	4, 16	1, 19	5, 13	3, 18
15	5, 19	1, 8	3, 8	2, 14	4, 9
16	1, 16	5, 17	4, 13	3, 17	2, 10
17	4, 15	1, 15	3, 8	2, 10	5, 20
18	5, 15	3, 13	2, 15	4, 9	1, 17
19	4, 8	3, 8	1, 15	5, 15	2, 6
20	2, 13	3, 9	1, 9	5, 13	4, 19

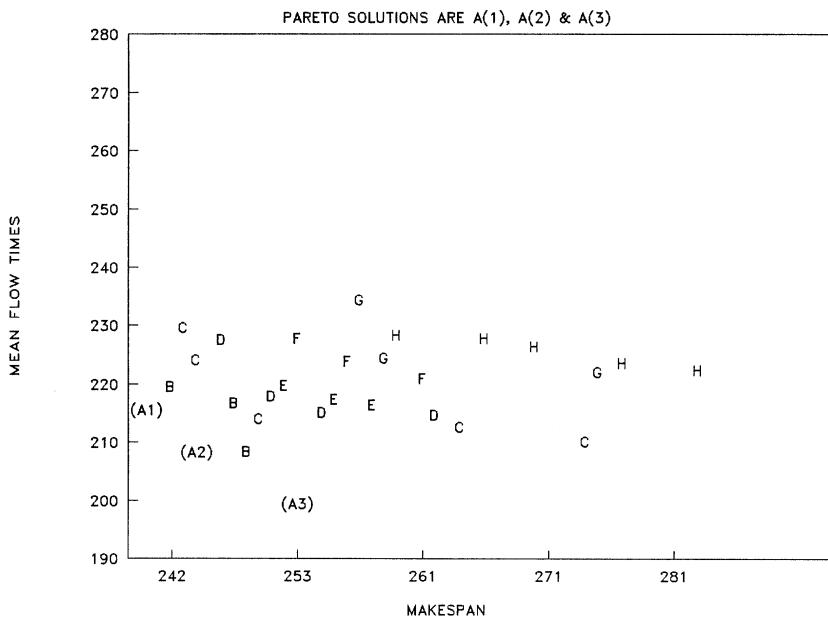


FIGURE 13.5 PARETO OPTIMAL AND SOME DOMINATED SOLUTIONS TO THE 10-JOB 10-m/c OPEN SHOP OF TABLE 13.4

TABLE 13.4 PROCESSING TIMES FOR A 10-JOB 10-m/c OPEN SHOP

	(m/c, t)									
Job 1	2, 13	9, 16	1, 17	3, 7	6, 14	7, 19	4, 13	8, 9	10, 12	5, 20
2	7, 12	5, 19	10, 5	9, 6	8, 15	4, 15	6, , 12	2, 8	1, 6	3, 12
3	8, 8	7, 8	6, 10	9, 20	4, 12	1, 10	3, 15	5, 15	10, 5	2, 17
4	3, 8	10, 20	4, 14	2, 19	8, 17	6, 18	5, 9	9, 20	7, 6	1, 8
5	5, 20	1, 13	3, 12	8, 7	9, 9	7, 10	2, 8	10, 16	4, 6	6, 16
6	8, 6	3, 14	10, 19	4, 14	9, 20	2, 12	1, 16	6, 17	7, 16	5, 6
7	2, 16	3, 6	8, 7	10, 17	7, 13	1, 17	5, 17	6, 17	4, 14	9, 15
8	4, 6	6, 13	3, 14	1, 9	10, 13	5, 13	7, 19	2, 15	8, 18	9, 16
9	7, 8	2, 13	3, 13	9, 7	10, 9	5, 10	6, 16	1, 11	4, 16	8, 18
10	3, 13	4, 18	1, 19	5, 6	10, 5	9, 19	2, 19	6, 6	8, 9	7, 18

PARETO OPTIMAL SCHEDULES FOR THE 10-M/C 10-JOB BI-OBJECTIVE OPEN SHOP PROBLEM SHOWN IN TABLE 13.4

SCHEDULE A(1): MAKESPAN: 242, MEAN FLOWTIME: 215.7

Operation #	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Sequence	67	34	85	89	11	5	43	73	98	19	17	16	99	80	44

Operation #	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
Sequence	1	68	22	57	58	50	56	31	81	10	29	39	62	94	32

Operation #	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45
Sequence	90	46	3	35	69	40	18	60	72	2	97	36	95	83	33

Operation #	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60
Sequence	8	77	9	93	65	24	47	64	82	91	74	45	41	25	21

Operation #	61	62	63	64	65	66	67	68	69	70	71	72	73	74	75
Sequence	96	30	66	55	87	86	4	26	37	7	13	70	79	49	71

Operation #	76	77	78	79	80	81	82	83	84	85	86	87	88	89	90
Sequence	20	42	12	27	63	78	51	15	100	53	84	92	14	38	52

Operation #	91	92	93	94	95	96	97	98	99	100
Sequence	61	48	75	64	59	54	88	28	23	76

SCHEDULE A(2): MAKESPAN: 247, MEAN FLOWTIME: 208.4

Operation #	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Sequence	44	1	68	22	57	58	34	50	56	81	31	10	29	39	89

Operation #	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
Sequence	62	69	46	98	79	24	37	92	25	82	85	16	95	3	70

Operation #	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45
Sequence	67	11	18	5	43	73	19	99	17	80	94	32	90	47	51

Operation #	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60
Sequence	78	35	15	7	100	4	53	65	6	84	87	13	49	93	45

SCHEDULE A(2) Continued.

Operation #	61	62	63	64	65	66	67	68	69	70	71	72	73	74	75
Sequence	41	21	96	30	55	66	71	20	40	91	42	74	9	12	60

Operation #	76	77	78	79	80	81	82	83	84	85	86	87	88	89	90
Sequence	27	97	86	63	72	2	36	83	33	8	77	14	26	38	52

Operation #	91	92	93	94	95	96	97	98	99	100
Sequence	61	48	75	64	59	54	88	28	23	76

SCHEDULE A(3): MAKESPAN: 255, MEAN FLOWTIME: 199.5

Operation #	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Sequence	67	85	34	89	11	5	43	73	98	19	17	16	80	99	29

Operation #	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
Sequence	94	32	90	46	1	3	44	35	68	81	69	40	18	60	72

Operation #	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45
Sequence	2	97	36	95	83	33	8	9	77	93	31	65	24	47	64

Operation #	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60
Sequence	82	49	27	42	12	41	62	75	91	74	22	25	50	86	59

Operation #	61	62	63	64	65	66	67	68	69	70	71	72	73	74	75
Sequence	53	57	10	38	63	51	70	45	7	56	87	13	79	39	71

Operation #	76	77	78	79	80	81	82	83	84	85	86	87	88	89	90
Sequence	20	78	15	100	4	84	6	58	37	92	88	30	28	21	96

Operation #	91	92	93	94	95	96	97	98	99	100
Sequence	55	66	14	26	52	54	23	76	48	61

TABLE 13.5 PROCESSING TIMES FOR A 20-JOB 10-m/c OPEN SHOP

	(m/c, t)									
Job 1	2, 13	9, 16	1, 17	3, 7	6, 14	7, 19	4, 13	8, 9	10, 12	5, 20
2	7, 12	5, 19	10, 5	9, 6	8, 15	4, 15	6, , 12	2, 8	1, 6	3, 12
3	8, 8	7, 8	6, 10	9, 20	4, 12	1, 10	3, 15	5, 15	10, 5	2, 17
4	3, 8	10, 20	4, 14	2, 19	8, 17	6, 18	5, 9	9, 20	7, 6	1, 8
5	5, 20	1, 13	3, 12	8, 7	9, 9	7, 10	2, 8	10, 16	4, 6	6, 16
6	8, 6	3, 14	10, 19	4, 14	9, 20	2, 12	1, 16	6, 17	7, 16	5, 6
7	2, 16	3, 6	8, 7	10, 17	7, 13	1, 17	5, 17	6, 17	4, 14	9, 15
8	4, 6	6, 13	3, 14	1, 9	10, 13	5, 13	7, 19	2, 15	8, 18	9, 16
9	7, 8	2, 13	3, 13	9, 7	10, 9	5, 10	6, 16	1, 11	4, 16	8, 18
10	3, 13	4, 18	1, 19	5, 6	10, 5	9, 19	2, 19	6, 6	8, 9	7, 18
11	1, 18	3, 15	8, 12	6, 8	7, 14	10, 11	2, 10	9, 11	5, 12	4, 14
12	6, 20	5, 13	3, 19	10, 15	7, 15	1, 16	9, 13	8, 8	4, 6	2, 12
13	3, 5	1, 20	2, 20	7, 10	4, 11	10, 19	6, 11	5, 14	8, 10	9, 9
14	7, 5	3, 13	10, 16	1, 20	4, 19	9, 16	6, 20	8, 8	5, 9	2, 18
15	2, 5	8, 6	6, 12	4, 15	1, 12	7, 8	3, 13	5, 10	10, 6	9, 10
16	4, 17	5, 12	10, 18	3, 20	7, 8	2, 9	8, 6	1, 19	6, 11	9, 18
17	9, 13	5, 5	2, 9	7, 12	3, 5	8, 7	1, 18	6, 9	4, 13	10, 19
18	5, 13	4, 12	9, 9	2, 19	10, 6	7, 20	6, 12	8, 5	3, 14	1, 10
19	8, 10	9, 11	10, 14	4, 10	3, 18	6, 11	2, 12	7, 19	1, 20	5, 12
20	8, 8	6, 18	9, 11	4, 18	7, 14	5, 7	10, 17	1, 5	3, 5	2, 18

EPILOG AND DIRECTIONS FOR FURTHER WORK

Multi-objective decision making is the typical musing in shop and classroom scheduling, in planning the route for the travelling salesman (even if we do not yet know how to tackle the problem), and in many other problems in management science formulated as combinatorial optimization. This text has described a practical methodology applicable to many such problems where finding Pareto optimal solutions may be of high value.

Specifically, the text gives you methods that can quickly produce Pareto solutions to flow, job and open shop scheduling problems involving 20-25 machines and 50-odd jobs. Finding nondominating solutions to such problems by conventional methods is not easy. By the methods described here, however, a bevy of such solutions may be found in a couple of minutes on a 400 MHz personal computer.

Based on its contents you would correctly infer that this book has not probed the theoretical structure of multi-objective shop scheduling problems. Thus, it has perhaps not advanced an understanding of the fundamental issues involved in solving such problems, whether or not that understanding leads to immediate applications. We wish to clarify that this book has deliberately chosen to walk the *applied* path—to help overcome specific difficulties in arriving at the *practical* solutions to such complex problems. In doing so it has anchored on the Pareto optimality rationality, borrowed know-how heavily from basic research on GAs and exploited recent innovations such as niching and fitness sharing.

However, even if it may have made multi-objective scheduling more accessible, this text has turned only a page of the potentially vast tome on the subject. It only shows a way where one will not have to worry about choosing "correct weights" in solving multi-objective problems

by weighted-sum type optimization. Still, ENGA-like methods lead to new questions. GA is very much empirical and it still misses the benefits of a sound theory. In the following pages we highlight some of the areas deserving deeper exploration.

- **Exact Solutions**

Branch-and-bound or dynamic programming (DP) can produce exact solutions to NP-hard scheduling problems. But such methods are usable only when the problem size is small. Exact methods, nevertheless, extend many benefits. Where practicable, these methods can be used to produce the *best* one-objective solutions—each objective taken one at a time—to help create benchmarks for solutions produced by a meta-heuristic. Therefore, notwithstanding the charm of meta-heuristic methods, research to seek exact solutions must continue.

- **Solving the General Job Shop**

Fox in Zweben and Fox (1994) describes a software called ISIS that is designed to tackle a complex job shop problem with multiple goals and constraints. The problem arose in General Electric in their turbine blades machining shop. ISIS used a framework in which certain constraints reflecting the physical realities and goals of the shop would be "resolved" (satisfied as closely as possible) by searching the solution space. Fox notes that the problem is NP-complete and describes a way to selectively relax the constraints, a method that he calls "constraint-directed search." Fox illustrates the use of beam search in the process.

However, from what we now know about the utility of GAs, problems solved by ISIS may be couched comfortably in a suitable chromosome structure that uses knowledge of the shop environment (see, for instance, Uckun, Bagchi, Kawamura and Miyabe, 1993). Thus, rather than attempting to seek compromises among the various constraints (objectives) attempted by ISIS, such representation would open the door to finding *nondominant* solutions to Fox's multi-goal JSP and, more broadly, the general JSP. Such rich chromosome structures and the associated genetic operators have not been touched upon in the present text.

- **Seeking Pareto Optimality**

At the risk of compelling a bias, we echo Zeleny's assertion that the Pareto optimality route is always a good path to follow in multi-objective decision making (Zeleny, 1985). However, in the past it has not been easy to *find* Pareto solutions. The difference that ENGA-like meta-heuristic methods make today is that now it is *possible* to find such solutions with only a modest effort. Conventionally, such problems are solved through the analytical aggregation of the different objectives into a single function.

One prefers Pareto solutions for some good reasons. The simultaneous optimization of multiple objectives seldom produces a single "Utopian" solution because the objectives are often in conflict or in competition. In some sense, each Pareto optimal nondominated solution may be likened to a saddle point in the decision space. The Pareto approach, if it is workable, finds a *family* of such points or alternative solutions that must be considered equivalent in the absence of information about each problem objective relative to the others. *Given* the nondominated solutions, the decision-maker needs only to select the "best" solution that meets some *other* preference criteria, such as minimum cost. Thus, any other (dominated) solution that is of lesser value to the decision-maker can be kept out of consideration.

- **Optimization of GA Parameters**

Techniques such as gradient-based methods and simplex-like algorithms as well as meta-heuristic methods such as simulated annealing or tabu search are difficult to extend as such to finding Pareto solutions. This is because these methods were not designed with multiple solutions in mind. On the other hand, because GAs work with a *population* of solutions, they are "naturally" suited to assemble Pareto solutions. GAs can also handle discontinuities, multimodality, disjoint feasible spaces as well as noisy function evaluations quite easily. Still, GA has no theory backing it. Importantly, no one has as yet satisfactorily explained why the GA process converges when properly parameterized, but not otherwise.

Thus, as of this writing, we use GA because "it works," and such evidence is still only empirical. Neither the schema theorem nor Markov chain-type GA models can predict its convergence behavior

in an arbitrary fitness landscape. In fact, the early theoretical studies of GAs produced results that did not bear out in practice (see Muhlenbein, 1997 page 144). The reason for such a predicament is that the stochastic process that develops when the GA executes is quite difficult to model. A potentially fertile approach, the application of population genetics to study solution evolution by GA, is in the early stage of development. Those interested in reading further should refer to Muhlenbein (1997).

Till GA *theory* evolves sufficiently, however, experimentation will continue to be the mainstay of *GA parameterization*. The results thus obtained are problem-specific as there is some interaction between GA parameters and the nature of the fitness landscape. As noted in Chapter 3, the statistical design-of-experiments (DOE) approach using pilot GA runs provides a robust and efficient framework to optimize GA parameters. In our experience, DOE can yield a robust GA that will display good on-line as well as off-line convergence regardless of the starting population. We also find that orthogonal arrays or partial factorial DOE designs incorporating up to three levels of each parameter suffice in most cases. Nevertheless, better methods may be sought to make GA parameterization even more efficient.

Apart from parameterization, a suitable chromosome representation (coding) of the problem being optimized must be constructed in order for mutation and crossover to create feasible offspring with high probability. Our experiments with timetabling and classroom scheduling suggest that the constraints of the problem should be coded *into* the chromosome structure as often as possible. This is better than seeking to handle the problem through penalty functions, etc. Muhlenbein (1997) provides a guideline here in which he maximizes the product of "heritability" and variance. But this too leads to problem-specific answers and deserves further study.

Many of these areas need to be probed deeper if GA has to computationally outdistance simulated annealing or tabu search. For instance, biologists have noted that all natural systems use identical genetic mechanisms based on the interaction between DNA and RNA. Is it possible to come up with a universal genetic representation for GAs, at least for combinatorial optimization problems? Another approach deserving attention is the use of small populations (see "micro GA" by Krishnakumar, 1989) to gain efficiency. Yet another way of obtaining efficient convergence is to use haploid (two-

chromosome) solutions and crossovers. Many organisms found in nature are haploid.

Nevertheless, we note that GAs are the only known approach at present that *naturally* produce a drove of Pareto optimal solutions to multi-objective problems. And, they do it in *one single run* of the algorithm.

- **ENGA vs. Other Multi-Objective Solution Methods**

Among the early methods used to *aggregate* multiple objectives (to enable single objective optimization methods to be applied to such problems) was the weighted-sum approach. Though convenient, the weighted-sum method often forces an artificial combination of objectives. Another method, known as goal attainment, minimizes the weighted difference between objective values and the corresponding goals. Such goals may be determined after the problem is solved as a single objective problem. Multiple attribute utility analysis has also been suggested for such problems, but without experimental results (Horn and Nafpliotis, 1993). Aggregation using penalty function is yet another approach suggested in the literature (e.g., Richardson, Palmer, Liepins and Hillard, 1989). However, all these methods are non-Pareto and they do not guarantee nondomination of the final solution with respect to the multiple objectives of the original problem. Among population-based *non-Pareto* approaches are the vector evaluated genetic algorithm (VEGA), and a method that uses lexicographic ordering.

VEGA selection adaptively attempts to balance improvement in several objective dimensions (Schaffer, 1985). VEGA treats non-commensurable objectives separately and searches for multiple non-dominated solutions concurrently in a single GA run. However, even though it identifies the nondominating individuals by monitoring the population, VEGA itself does not use this information. Further, VEGA causes undesirable clustering of solutions into *species* (solutions particularly strong in some individual objective) when concave trade-off surfaces are involved, because VEGA implicitly uses a linear combination of the objectives (Fleming and Pashkevich, 1985).

Pareto domination-based fitness assignment incorporated in NSGA (Chapter 8) as well as in ENGA (Chapter 10) was first proposed by Goldberg (1989). This method assigns equal probability of

reproduction (during selection) to all non-dominated individuals in the population and less to others. Fitness here is not determined directly by the actual objectives being optimized, but rather by the Pareto rank of a solution. Several different ranking schemes based on Pareto dominance have since been proposed. Fonseca and Fleming (1993) suggest that the individual's rank should correspond to the number of individuals in the current population by which it is dominated. Nondominated individuals are therefore all assigned the same rank while dominated ones are penalized according to population density in the corresponding region of the trade-off surface. Selection in the Fonseca-Fleming scheme is by stochastic universal sampling (Baker, 1987). In the Horn-Nafpliotis scheme (Horn and Nafpliotis, 1993), selection is tournament type, based on Pareto dominance.

Pareto ranking of the kind used by Fonseca and Fleming (1993) and by Srinivas and Deb (1995, the creators of NSGA), offers several advantages. First, the method is oblivious to the convexity or non-convexity of the trade-off surface. In this way it attenuates the proneness of the solutions to speciation. The method also rewards good performance in any objective dimension regardless of the others and solutions that exhibit good performance in many, if not all, objectives, are more likely to be produced by recombination. However, Pareto ranking, by itself, does not guarantee that the Pareto set would be uniformly sampled. This is desirable when we are seeking a family of solutions *well distributed* over the Pareto front.

One way to spread the solutions over the Pareto optimal front or the "efficient front" (Figure 6.1) is to do *fitness sharing*. Fitness sharing prevents genetic drift (a phenomenon occurring in small size populations undergoing evolution in which chance deviations cause changes in allelic frequency, see Goldberg, 1989, and Russell, 1998). Good guidelines are now available to estimate the niche sizes within which sharing is likely to occur. Fonseca and Fleming (1993), Horn and Nafpliotis (1993) and Cieniawski (1993) have all recorded superior performance of GA schemes in achieving diversity when sharing is done.

Srinivas and Deb (1995) perform sharing in the decision variable domain. This enables multiple solutions with nearly identical objective function values to form and the selection mechanism is truly independent of objective scaling. This method of sharing is

incorporated in both NSGA (Chapter 8) and ENGA (Chapter 10) and demonstrated in this text. Nevertheless, further understanding of the effect of different sharing mechanisms would be of high value.

Niching and speciation produce stable populations of diverse yet co-existing life forms in nature. Actively promoting niche formation is a key strategy used in both NSGA and ENGA. This produces multiple Pareto optimal solutions that are well spread along the Pareto front. But, the optimum exploitation of niching and speciation is presently guided empirically at best. It may be worth studying these two phenomena using population genetics-type methods.

- **Conflicting and Synergistic Optimization Objectives**

We turn now to another important question. What does ENGA do when some or all the objectives being simultaneously optimized are *not* in conflict? "Conflicting objectives" is a condition that we stated as the "requirement" to qualify a problem to be of multi-objective decision making or MCDM (multi-criteria decision making) class (Section 6.2). The point is that if the objectives are not in conflict, the problem may be solved by optimizing *any one* of the objectives, by any single objective optimization method. Otherwise, multi-criteria methods would have to be used. Perhaps other questions may be asked: Is it necessary for the analyst to *anticipate* objective conflicts among the multiple objectives he/she is attempting to optimize? Given several objectives with *unknown* relationships among them, does he/she need to take some special steps before applying, say, ENGA? This question is empirically answered by Rao (1999), who used ENGA to guide the optimization of a 3-objective injection molding process.

To conserve experimental effort, labor, utilities and molding material (because each GA generation requires the determination of the fitness of *each* solution in the population), Rao first produced three separate second order response models (14.1-14.3) for the three responses of interest. Thus, actual molding runs were made only to develop the response models. Subsequently, he used these models as surrogates of the actual process to evaluate the responses under specified conditions as and when required by ENGA. (At the end of the study *verification runs* established the final solutions' quality and their merit.)

The multiple regression models developed for a part "diskcady" to be molded using polystyrene material are shown by (14.1), (14.2) and (14.3). Process conditions—mold temperature (X_1), melt temperature (X_2) and injection time set (X_3)—were the independent or "regressor" variables. The three responses were injection pressure, actual injection time and temperature difference. Once these models had been built, response evaluations as called for by GA were done by using these surrogates of the process, rather than by running the actual molding process.

$$\begin{aligned} \text{INJECTION PRESSURE} = & 288.139 - 0.15742*X_1 - \\ & 1.45338*X_2 - 66.66*X_3 - 0.00065*X_1^2 + \\ & 0.001644*X_2^2 + 7.968747*X_3^2 + 0.000945*X_1*X_2 \\ & - 0.04881*X_1*X_3 + 0.171098*X_2*X_3 \end{aligned} \quad (14.1)$$

$$\begin{aligned} \text{ACTUAL INJECTION TIME} = & 0.247042 - 0.00019*X_1 - \\ & 0.00235*X_2 + 1.06185*X_3 - 1.8E-06*X_1^2 + \\ & 5.56E-06*X_2^2 - 0.0016*X_3^2 + 1.67E-06*X_1*X_2 \\ & + 6.68E-05*X_1*X_3 + 2.22E-15*X_2*X_3 \end{aligned} \quad (14.2)$$

$$\begin{aligned} \text{TEMPERATURE DIFFERENCE} = & -5.87416 + 0.018823*X_1 + \\ & 0.05786*X_2 + 1.36318*X_3 - 5.5E-05*X_1^2 - \\ & 0.00013*X_2^2 - 0.65733*X_3^2 - 8.8E-05*X_1*X_2 - \\ & 0.01985*X_1*X_3 + 0.02146*X_2*X_3 \end{aligned} \quad (14.3)$$

To summarize, it is *not* necessary to establish that the different objectives *are* in conflict before one applies ENGA. If the objectives are in conflict (as indicated by Figures 14.1A and 14.2A), ENGA will produce the Pareto front containing the efficient solutions. If they are not (as in Figure 14.3A), ENGA will converge on its own accord to the *Utopian* (all objectives optimized at one point) solution. This is so because Pareto-based selection finds Utopian solutions also, if they exist (Fonseca and Fleming, 1995). This rarely happens in weighted-sum multi-objective optimization.

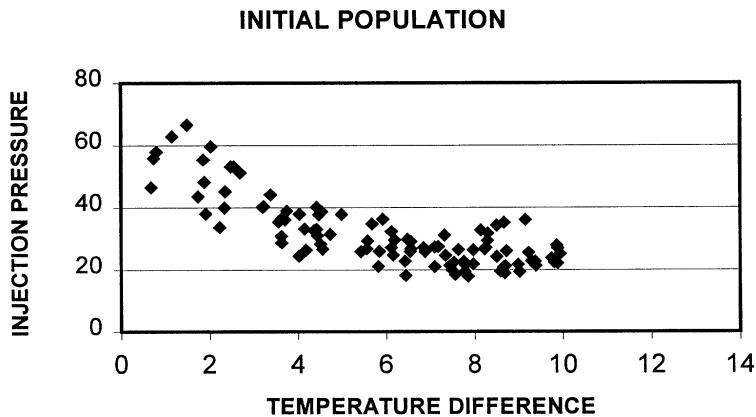


FIGURE 14.1A SCATTER PLOT OF TEMPERATURE DIFFERENCE vs. INJECTION PRESSURE, INITIAL RANDOM POPULATION

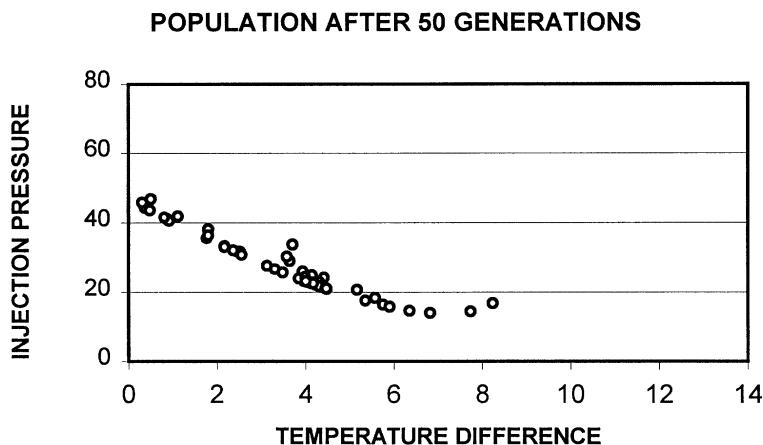
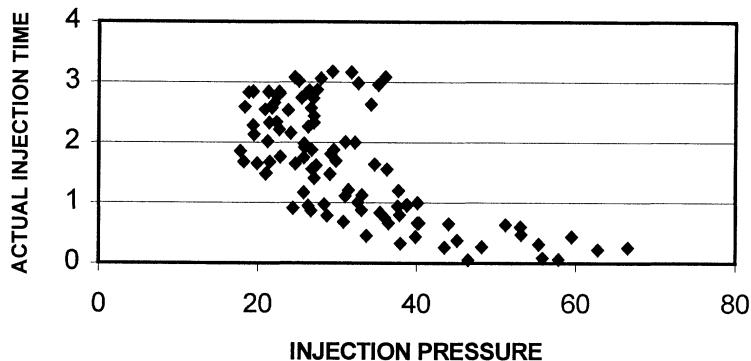


FIGURE 14.1B SCATTER PLOT OF TEMPERATURE DIFFERENCE vs. INJECTION PRESSURE AFTER 50 GENERATIONS OF ENGA

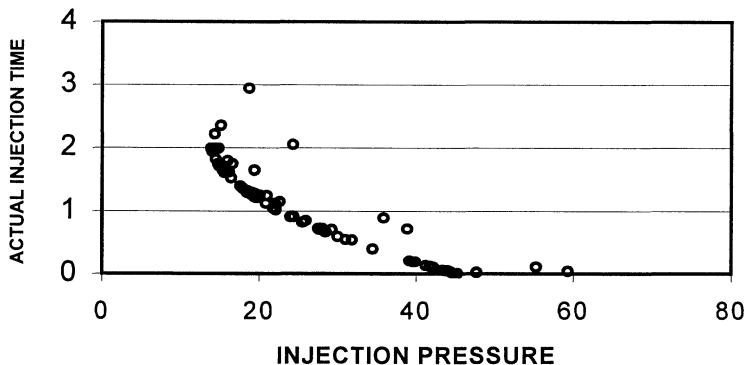
The scatter plot in Figure 14.1A shows that in general, responses injection pressure and temperature difference (between the extremities of the part produced) have a negative correlation and therefore are in conflict. When bi-objective minimization is sought, ENGA converges here toward the Pareto front shown in Figure 14.1B.

INITIAL POPULATION



**FIGURE 14.2A SCATTER PLOT OF INJECTION PRESSURE vs.
ACTUAL INJECTION TIME, INITIAL RANDOM POPULATION**

POPULATION AFTER 50 GENERATIONS



**FIGURE 14.2B SCATTER PLOT OF INJECTION PRESSURE VS.
ACTUAL INJECTION TIME, AFTER 50 GENERATIONS OF ENGA**

Responses injection pressure (applied) and actual injection time are in conflict as hinted by Figure 14.2A. When bi-objective minimization is sought, ENGA converges here toward the Pareto optimal front as shown in Figure 14.2B.

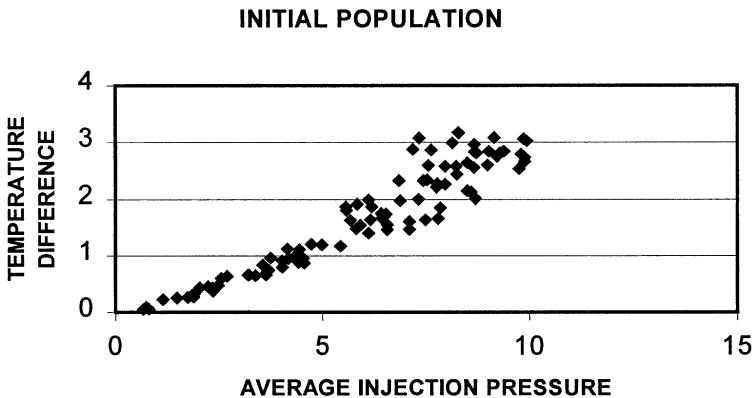


FIGURE 14.3A SCATTER PLOT OF AVERAGE INJECTION PRESSURE vs. TEMPERATURE DIFFERENCE, INITIAL RANDOM POPULATION

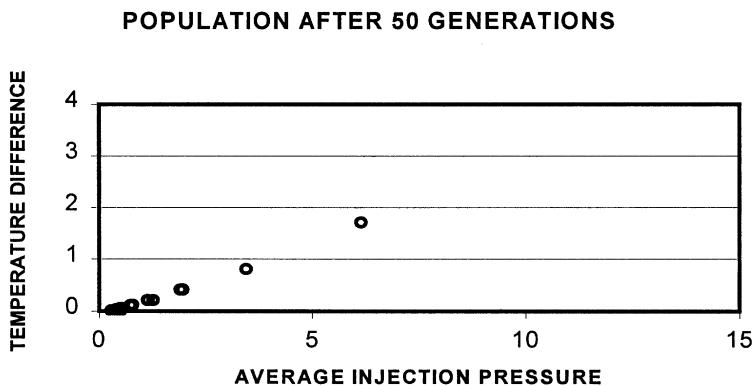


FIGURE 14.3B SCATTER PLOT OF AVERAGE INJECTION PRESSURE VS. TEMPERATURE DIFFERENCE, AFTER 50 GENERATIONS OF ENGA

Responses injection pressure and temperature difference (between the extremities of the parts produced) are not in conflict as hinted by Figure 14.3A. When bi-objective minimization is sought, ENGA converges here toward the left, lower corner of the plot (the Utopian solution) as shown in Figure 14.3B.

- **Darwinian and Lamarckian GAs: The high value of hybridizing the GA**

Chapter 4 set out a synopsis of the common algorithms and heuristics used to solve the m -machine n -job flowshop scheduling problem, a problem that is NP-hard (Pinedo, 1995). The heuristics reviewed included Palmer's heuristic, Gupta's heuristic, the CDS heuristic, the RA heuristic and the NEH heuristic. Chapter 4 also sketched several published methods for applying genetic algorithms to sequence the flowshop. The different GA-based and hybridized approaches were recalled. Subsequently, issues uncovered by researchers about starting solution choice, the choice of GA parameters and certain merits of hybridization were noted.

We find good support for the following assertions about using GA to sequence flowshops:

- Given time, the Darwinian GA generally produces solutions superior to those produced by solution-generating heuristics such as Palmer, CDS, NEH or RA applied alone.
- However, for every *strategic* combination of solution methods tested (as indicated by Table 4.4), the pure Darwinian GA operating alone and started with a random initial population yields a statistically larger makespan.
- Whenever the initial population is given a "kick start" by filling it up with all-identical Palmer, CDS, NEH or the RA heuristic solutions, results improve.
- When Lamarckism is introduced by improving the best solution by "compressing" it, by invoking the Ho-Chang improvement heuristic, results improve further.
- An essential constituent in this process is the use of mutation. This enables the process to avoid getting prematurely stuck.
- In almost every learning-incorporated (i.e., hybridized) case we should expect to produce solutions better than what is produced by random search or by the stand-alone application of the solution generating heuristic methods—Palmer, NEH, CDS and RA.

However, even if we recommend the use of learning-incorporated GAs, we do not for a minute downplay the value of *independently* developing solution-generating or solution-improving heuristics. Indeed both are highly valuable in speeding up genetic algorithms. Solution-generating heuristic methods "heat up the engine block"

quite effectively so as to assure a running start while solution-improvement methods such as the Ho-Chang heuristic jump start the GA when it appears to be "puttering."

Therefore, we strongly endorse continued research for both good solution-generating *as well as* solution-improving heuristic methods. We also recommend the development of innovative ways to incorporate Lamarckism, sharing, etc. to augment the effectiveness of the conventional (Darwinian) GA in shop scheduling.

For multi-objective scheduling problems hybridizing may be used as follows. After mutation and crossover have been applied, the solutions may be locally climbed with respect to each objective where possible, although we need not improve the *same* chromosome by more than one type of local climb. Some experimentation would be needed here to make the procedure work well. We expect that the gains made by local climbs on several objectives will interject good genes (building blocks) into the schedules on hand. This would improve both the GA's convergence rate as well as the quality of the solution.

- **Concluding Remarks**

We began this text with a review of the objectives of scheduling and of the published literature on single-objective and multi-objective flowshop scheduling. We then appraised the utility of NSGA, a meta-heuristic method proposed by Srinivas and Deb (1995) that produces Pareto optimal solutions.

We used statistically designed experiments in the parameterization of the particular NSGA forms constructed in order to best tackle the flowshop. We noted that NSGA *does not* preserve good solutions (the already "efficient" schedules) from generation to generation. Thus, the good (near-optimal) solutions lost in one generation by mutation or recombination have only a probabilistic chance in NSGA to reappear in a later generation. Also, the *number* of final solutions (efficient schedules) on the Pareto optimal front (i.e. the distribution of distinct solutions on the front) in NSGA remained comparatively low. This was true even with the DOE-guided "best" parameterization applied and after a large number of GA generations.

To deal with these limitations, an enhancement for NSGA was constructed, a procedure that we dubbed ENGA (the Elitist Nondominated Sorting GA). ENGA uses one *additional* nondominated sorting and Pareto ranking of the combined parents + progeny population (Figure 10.1). Subsequently it was statistically determined that ENGA performed better than NSGA on the problems tested.

NSGA and ENGA were then compared using the *multi-objective job shop* and the *multi-objective open shop*. Here again ENGA outperformed NSGA. We record that on an average, ENGA took about two minutes to produce 100 Pareto optimal solutions to a three-objective flowshop problem involving 49 jobs and 15 machines. This algorithm was implemented in compiled C++ on a 400 MHz personal computer.

In summary, while many other variations are possible, this new metaheuristic method appears to tackle multi-objective flowshop, job shop and open shop problems quite effectively. It will also solve single-objective scheduling more efficiently because it uses *sharing*, an operation not used in SGA. Note that no method—analytical or heuristic—currently exists to tackle these challenging yet ubiquitous scheduling problems.

A number of extensions of the material described in this text are possible. They are the following.

Srinivas and Deb (1995) remark that NSGA parameters *population size* and σ_{share} play an important role in controlling the effectiveness of nondominating sorting. The population size requirement may be larger when the number of objectives being simultaneously optimized increases. But how this requirement would increase is a subject for future research.

In the implementation of ENGA to JSP we used an *operation-based representation* for the chromosomes. Other representations such as machine-based representation discussed in Chapter 3 may also be investigated for their relative efficacy. Such rich representations are expected to help one tackle the general JSP.

Keeping population size constant is a prevailing tradition in GA. This puts an artificial upper limit on the number of Pareto efficient solutions found by NSGA/ENGA. A possible variation in ENGA

may be the retention of *only and all* the Front 1 members, that are possibly reproduced or deleted randomly to build up a target population size if necessary, since all Front 1 members have truly the same qualification ("nondomination fitness") and each member is an *efficient solution* to the multiobjective problem. Such ideas may be empirically studied.

Another variation to ENGA may be explored by limiting the maximum population size to some practical limit and by retaining (transferring to the next generation) only the members belonging to Front 1.

A general purpose scheduling software may be developed based on ENGA. Such software would use sharing and handle both single and multi-objective JSP. Then it would help ensure maximum use of resources in a hospital or airport or a shop requiring *high flexibility* (as in FMS) and *rapid response* to changing demand, processing and due date requirements. We also envisage ENGA extensions to stochastic scheduling, including machine failures and planned downtimes, for the GA can be easily interfaced with simulation.

The lesson learned from the methods explored in this text is that in evolutionary computing one need not blindly copy nature, for it is not entirely clear that mother nature is trying to optimize anything. Rather, as perceptively pointed out by Darwin (1860) and noted by Holland (1975) and Gould (1993), she aims at adaptation. On the other hand, management science will gain much more in being inspired by the sheer ingenuity and enormity of life processes. As hinted by the superior performance of hybridized GAs in shop scheduling and TSP problems, a great many tools and implements (the analytical and heuristic methods being continually created) can be quite easily adapted to evolutionary computing, thus impacting on the effectiveness and utility of both.

In the Appendix we have included some Boreland Turbo C++® (Version 3) codes originally developed by Jain (1999) for single objective flowshop scheduling. This might enable you to experiment with genetic algorithms—if you haven't done so already. The applets in Sections 10.2 and 10.3 are intended to help you begin probing the frontiers of multi-objective GAs.

References

- Aarts E and J K Lenstra (1997). *Local Search in Combinatorial Optimization*, John Wiley.
- Adams J, E Balas and D Zawack (1988). The Shifting Bottleneck procedure for Jobshop Scheduling, *Int'l J. of Flexible Manufacturing Systems*, 34-3, 391-401.
- Adulbhan P and M T Tabucanon (1980). Multicriterion Optimization in Industrial Systems. Chapter 9, *Decision Models for Industrial Systems Engineers and Managers*, Asian Institute of Technology, Bangkok.
- Anderson E J, C A Glass and C N Potts (1997). Machine Scheduling in Aarts and Lenstra (1997), 361-414.
- Applegate D and W Cook (1991). A Computational Study of the Jobshop Scheduling Problem, *ORSA J. of Computing*, 3-2, 149-156.
- Bagchi Tapan P (1993). *Taguchi Methods Explained: Practical Steps to Robust Design*, Prentice-Hall (India).
- Bagchi Tapan P and C Sriskandarajah, (1996). A Genetic Algorithm to Minimize Makespan in a Blocking Flowshop, *Proceedings of the First APDSI Conference*, Hong Kong, June, 579-589.
- Bagchi Tapan P and J G C Templeton (1994). Multiple-criteria Robust Design using Constrained Optimization, *J. Design and Manufacturing*, 4, 21-30.
- Bagchi Tapan P and Kalyanmoy Deb (1996). Calibration of GA Parameters: The Design of Experiments Approach, *Computer Science and Informatics*, 26, 3.
- Bagchi Tapan P and Madhu Ranjan Kumar (1993). Multiple-Criteria Robust Design of Electronic Devices, *J. of Electronic Manufacturing*, 3, 1993, 31-38.
- Bagchi Tapan P, J G C Templeton and Krishan Raman (1995). Effectiveness of Evolutionary Searches in Robust Design, *TIMS XXXIII INFORMS Singapore Int'l Meeting*, June 27-29.
- Baker J E (1987). Reducing Bias and Inefficiency in the Selection Algorithm in Grefenstette (1987).
- Baker K (1974). *Introduction to Sequencing and Scheduling*, John Wiley & Sons.

- Baker K and G Scudder (1990). Sequencing with Earliness and Tardiness Penalties: A Review, *Operations Research*, 38, 22-36.
- Balas E (1969). Machine Sequencing via Disjuncting Graphs: An Implicit Enumeration Algorithm, *Operations Research*, 17, 941-957.
- Balas E and A Vazacopoulos (1994). Guided Local Search with Shifting Bottleneck for Job Shop Scheduling, *Management Science Research Report MSRR-609*, Carnegie Mellon University.
- Beale G O (1977) Optimal Aircraft Simulator Development by Adaptive Random Search Optimization, *Ph D Dissertation*, University of Virginia.
- Beale G O and G Cook (1978). Optimal Digital Simulation of Aircraft via Random Search Techniques, *J. Guidance and Control*, 1-4.
- Bean J (1994). Genetic Algorithms and Random keys for Sequencing and Optimization, *ORSA J. of Computing*, 6-2, 154-160.
- Bean J and B Norman (1995). Random keys Genetic Algorithm for Jobshop Scheduling: Unabridged version, University of Michigan.
- Belew R and L Booker (1992). *Proceedings of the Fourth Int'l Conference on Genetic Algorithms*, Morgan Kaufmann.
- Bhatnagar A (1996). Scheduling Jobs in the Generalized Jobshop, *M Tech Thesis*, Department of Industrial and Management Engineering, Indian Institute of Technology Kanpur.
- Birch L C (1957). The Meaning of Competition, *American Nature*, 91, 5-18.
- Blackstone J, D Phillips and G Hogg . (1982). A State-of-the-Art Survey of Dispatching Rules for Manufacturing Jobshop Operations, *Int'l J. of Production Research*, 20, 26-45.
- Blazewicz J, K Ecker, G Schmidt and J Weglarz, (1994). *Scheduling in Computer and Manufacturing Systems*, 2nd edition., Springer-Verlag.
- Box G E P (1957). Evolutionary Operation: A Method for Increasing Industrial Productivity, *Applied Statistics*, 6, 81-101.
- Bremermann H J, J Roghson and S Salaff (1966). Global Properties of Evolution Processes in H H Pattee, E A Edelsack, L Fein and A B Calahan (eds.) (1966) *Natural Automata and Useful Simulations*, Macmillan, 3-42.
- Bruns Ralf (1993). Direct Chromosome Representation and Advanced Genetic Operators for Production Scheduling, *Proceedings of the Fifth Int'l Conference on Genetic Algorithms*, 352-359.
- Buzacott John A and S George Shanthikumar (1992). *Stochastic Models of Manufacturing Systems*, Prentice-Hall.
- Campbell H, R Dudek, and M Smith (1970). A Heuristic Algorithm for the n -Job m -Machine Sequencing Problem, *Management Science*, 16B, 630-637.

- Carter M (1997). *Proceedings, 2nd Int'l Conference on the Practice and Theory of Automated Timetabling*, University of Toronto, Aug 20-22.
- Castillo E D and D C Montgomery (1993). A Nonlinear Programming Solution to the Dual Response Problem, *J. Quality Technology*, 25, 3, 199-204.
- Cavicchio D J (1970). Adaptive Search using Simulated Evolution, *Unpublished Doctoral Dissertation*, University of Michigan.
- Chakraborti C and K K N Sastry (1998). The Genetic Algorithms Approach for Proving Logical Arguments in Natural Language, *Genetic Programming-98*, John Koza et al. (eds.), Madison, WI, Morgan Kaufmann.
- Chen C L, R V Neppalli and N Aljaber (1996). Genetic Algorithms Applied to the Continuous Flowshop Problem, *Computers Ind. Eng.*, 30(4), 919-929.
- Chen C, V S Vempati and N Aljaber (1995). An Application of Genetic Algorithms for Flowshop Problems, *Eu. J. of Operational Research*, 80, 389-396.
- Cheng R, M Gen and Y Tsujimura (1996a). A Tutorial Survey of Jobshop Scheduling Problems using Genetic Algorithm: Part I Representation, *Int'l J. of Computers and Industrial Engineering*, 30-4, 983-997.
- Cheng R, M Gen and Y Tsujimura (1996b). A Tutorial Survey of Jobshop Scheduling problems using Genetic Algorithm: Part II, *Int'l J. of Computers and Industrial Engineering*, 30-4.
- Cheng T and C Sen (1990) A State-of-the-Art review of Parallel-Machine Scheduling Research, *Eu. J. of Operational Research*, 47, 271-292.
- Cieniawski S E (1993). An Investigation of the Ability of Genetic Algorithms to Generate the Tradeoff Curve of a Multi-objective Groundwater Monitoring Problem. *Master's Thesis*, University of Illinois at Urbana-Champaign.
- Cochran W G and G M Cox (1957). *Experimental Designs*, 2nd edition, Wiley.
- Cochrane J L and M Zeleny (eds.) (1973). *Multiple Criteria Decision Making*, University of South Carolina Press.
- Coffman E (1976). *Computer and Job-Shop Scheduling Theory*, John Wiley & Sons.
- Connolly D T (1992). General Purpose Simulated Annealing, *Operations Research*, 43, 3, 495-505.
- Conway R, W Maxwell and L Miller (1967). *Theory of Scheduling*, Addison-Wesley.

- Croce F, R Tadei and G Volta (1995). A Genetic Algorithm for the Jobshop Problem, *Computer and Operations Research*, 22, 15-24.
- Daniels R (1990). A Multi-Objective Approach to Resource Allocation in Single Machine Scheduling, *Eur. J. of Operational Research*, 48, 226-241.
- Dannenbring D (1977). An Evaluation of Flkowshop Sequencing Heuristics, *Management Science*, 23, 1174-1182.
- Darwin C (1860). *On the Origin of Species by means of Natural Selection, or the Preservation of Favoured Races in the Struggle for Life*, Appleton.
- Davidor Y (1991). *Genetic Algorithms and Robotics*, World Scientific.
- Davis L (1985). Jobshop scheduling with Genetic Algorithms, in Grefenstette (1985), 136-140.
- Davis L (1989). Adapting Operational Problems in GA, *Proceedings of the Third Int'l Conference on GAs*, San Mateo, CA.
- Davis L (ed.) (1991). *Handbook of Genetic Algorithms*, Van Nostrand.
- Dawkins Richard (1996). *Improbable Climbing Mount*, Penguin Books.
- De Jong K A (1975). An Analysis of the Behavior of a Class of Genetic Adaptive Systems, *Doctoral Dissertation*, University of Michigan.
- De Jong K A (1993). Genetic Algorithms are NOT Function Optimizers, *FOGA*, 5-17.
- De Robertis E D P and E M F De Robertis Jr. (1987). *Cell and Molecular Biology*, 8th edition, Info-Med Limited.
- Deb K (1993). Genetic Algorithms for Engineering Design Optimization, *Proceedings of the Advanced Study Institute on Computational Methods for Engineering Analysis and Design*, Madras, 12.1-12.25.
- Deb K (1995). *Optimization for Engineering Design: Algorithms and Concepts*, Prentice-Hall (India).
- Deb K and D E Goldberg (1989). An Investigation of Niche and Species Formation in Genetic Function Optimization, *Proceedings of the third Int'l Conference on GA*, 97-106, Morgan-Kaufmann.
- Dileepan P and T Sen (1988). Bicriterion Static Scheduling Research for a Single Machine, *Omega*, 16, 53-59.
- Dobzahansky Theodosius (1970). *Mankind Evolving*, Bantam Books.
- Dobzahansky Theodosius (1986). Evolution, *Encyclopedia Americana*, Grolier.
- Dorndorf U and E Pesch (1995). Evaluation Based Learning in a Jobshop Scheduling Environment, *Computers and Operations Research*, 22, 25-40.
- Dowsland K A (1996). Genetic Algorithms—A Tool For OR, *Operations Research*, 47, 550-561.

- Draper N R and H Smith (1998). *Applied Regression Analysis*, Wiley.
- Dudek R, S Panwalkar and M Smith (1992). The Lessons of Flow Shop Scheduling Research, *Operations Research*, 40, 7-13.
- Emmons Hamilton (1975). A Note on a Scheduling Problem with Dual Criteria, *Naval Research Logist. Quart.* 22, 615-616.
- Eshelman L J (ed.) (1995). *Proceedings of the Sixth Int'l Conference on Genetic Algorithms*, Morgan Kauffmann Publishers, San Francisco.
- Falkenauer E and S Bouffouix. (1991). A Genetic Algorithm for Job shop, Proceeding of the IEEE Int'l Conference on Robotics and Automation, 824-829.
- Fang H, P Ross and D Corne (1993). A Promising Genetic Algorithm Approach to Job-shop Scheduling, Rescheduling, and Open-shop Scheduling Problems, in Forrest (1993), 375-382.
- Flippone S F (1989). Using Taguchi Methods to Apply to Axioms of Design, *Robot, Computer Integrated Manufacturing*, 6(2), 133-142
- Fogarty T (ed.) (1994). *Evolutionary Computing*, Springer-Verlag, Berlin.
- Fogel D (ed.) (1994). *Proceedings of the First IEEE Conference on Evolutionary Computation*, IEEE Press.
- Fonseca Carlos M and Peter J Fleming (1993). Genetic Algorithms for Multiobjective Optimization: Formulation, Discussion and Generalization, In S. Forest (ed.) (1993), *Proceedings of the Fifth Int'l Conference on Genetic Algorithms*, 416-423.
- Fonseca Carlos M and Peter J Fleming (1995). An Overview of Evolutionary Algorithms in Multiobjective Optimization, *Evolutionary Computation*, 3(1), 1-16, Spring.
- Forrest S (ed.) (1993). *Proceedings of the Fifth Int'l Conference on Genetic Algorithms*, Morgan Kaufmann.
- French S (1982). Sequencing and Scheduling: *An Introduction to the Mathematics of the Job Shop*, Wiley.
- Garey M R and D S Johnson (1979). *Computers and Intractability--A Guide to the Theory of NP-Completeness*, W H Freeman.
- Garey M R, D S Johnson and R Sethi (1976). The Complexity of Flowhsop and Jobshop Scheduling, *Mathematics of Operations Research*, 1, 117-129.
- Gen M and R Cheng (1997). *Genetic Algorithm and Engineering Design*, John Wiley & Sons.
- Gen M and T Kobayashi (1994) *Proceedings of the 16th Int'l Conference on Computers and Industrial Engineering*, Ashikaga, Japan.

- Gen M, Y Tsujimura and E Kubota (1994). Solving Job-Shop Scheduling Problem Using Genetic Algorithms, in Gen and Kobayashi (1994), 576-579.
- Gibbons Jean D (1996). *Nonparametric Methods for Quantitative Analysis*, 3rd edition, American Sciences Press.
- Giffler B and G Thompson. (1960). Algorithms for Solving Production Scheduling Problems, *Operations Research*, 8-4, 487-503.
- Glover Fred W and M Laguna (1997) *Tabu Search*, Kluwer Academic.
- Goldberg D E (1985). Optimal Initial Population Size for Binary-coded Genetic Algorithms. *TCGA Report No. 85001*. Tuscaloosa: University of Alabama.
- Goldberg D E and Dirk Thierens, (1993a) Mixing in Genetic Algorithms, *Working Paper*, Dept. of General Eng., University of Illinois at Alabama.
- Goldberg D E and J J Richardson (1987). Genetic Algorithms with sharing for multi-modal function optimization, *Genetic Algorithms and Their Applications: Proceedings of the Second Int'l Conference on Genetic Algorithms*, Lawrence Erlbaum Associates, 41-49
- Goldberg D E, Dirk Thierens and K Deb (1993b). Toward a better understanding of Mixing in Genetic Algorithms. *Journal of SICE*, 32 (1).
- Goldberg G E (1989). *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison-Wesley.
- Gould Stephen Jay (1977). (1980) *The Panda's Thumb*, Norton
- Gould Stephen Jay (1993). *Eight Little Piggies: Reflections in Natural History*, Penguin.
- Gould Stephen Jay (1996). *The Spread of Excellence from Plato to Darwin*, Vintage.
- Greenberg H (1968). A Branch-and-Bound Solution to the General Scheduling Problem, *Operations Research*, 16, 353-361.
- Grefenstette J (1991). Lamarckian Learning in Multi-agent Environment, in the *Proceedings of the Fourth Conference on Genetic Algorithms*, Morgan-Kaufmann.
- Grefenstette J J (1983) *A User's guide to GENESIS*, Tech Report CS-83-11, Computer Science Department, Vanderbilt University.
- Grefenstette J J (1986). Optimization of Control Parameters for Genetic Algorithms, *IEEE Trans. on Systems, Man, and Cybernetics*, . 16 (1).
- Grefenstette J J (ed.) (1985). *Proceedings of the First Int'l Conference on Genetic Algorithms*, Lawrence Erlbaum Associates.

- Grefenstette J J (ed.) (1987). *Proceedings of the Second Int'l Conference on Genetic Algorithms and their Applications*, Lawrence Erlbaum Associates.
- Grefenstette J J and J E Barker, (1987). How GAs work: A Critical Look at Implicit Parallelism in Grefenstette J J (ed.), *Proceedings of the Second Int'l Conference on Genetic Algorithms*, 59-68.
- Grierson D E and Prabhat Hajela (ed.) (1996). Emergent Computing Methods in Engineering Design: Applications of GAs and Neural Networks, NATO ASI series, Series F: Computer and Systems Sciences, 149.
- Gupta J (1971). A Functional Heuristic Algorithm for the Flowshop Scheduling Problem, *Operations Research Quarterly*, 22, 39-47.
- Gupta S and J Kyparisis (1987). Single Machine Scheduling Research, *Omega*, 15, 207-227.
- Hancock P (1994). An Empirical Comparison of Selection Methods in Evolutionary Algorithms, in Fogarty (1994), 80-95.
- Hans A E (1988). Multicriteria Optimization for Highly Accurate Systems. In W. Stadler (ed.), Multicriteria Optimization in Engineering and Sciences, *Mathematical Concepts and Methods in Science and Engineering*, 19, 309-352.
- Haupt R (1989). A Survey of Priority Rule-based Scheduling, *OR Spectrum*, 11, 3-16.
- Heck H and S Roberts (1972). A note on the Extension of a Result on Scheduling with a Secondary Criteria, *Nav. Res. Log. Quarterly*, 19, 403-405.
- Ho J C and Y L Chang (1991). A New Heuristic for the n -Job, m -Machine Flowshop Problem, *Eur. J. of Operational Research*, 52, 194-202.
- Holland J H (1972). *Adaptation in Natural and Artificial Systems: An introductory Analysis with Application to Biology, Control, and Artificial Intelligence*, MIT Press, Cambridge, MA.
- Holland J H (1975). *Adaptation in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor.
- Holland J H (1992). Genetic Algorithms, *Scientific American*, 15-21.
- Holsapple C, V Jacob, R Pakath and J Zaveri (1993). A Genetics-based Hybrid Scheduler for generating Static Schedules in Flexible Manufacturing Contexts, *IEEE Transactions on Systems, Man and Cybernetics*, 23, 953-971.
- Horn J and N Nafpliotis (1993). Multi-objective Optimization using the Niched Pareto Genetic Algorithm, *Illigal Report 93005*, University of Illinois at Urbana-Champaign.

- Horn J, N Nafpliotis and Goldberg D E (1994). A Nicched Pareto Genetic Algorithm for Multiobjective Optimization, *IEEE World Congress on Computational Intelligence*, 1, 82-87.
- Hundal T S and J Rajgopal (1988). An Extension of Palmer's Heuristic for the Flowshop Scheduling Problem, *Int'l J. of Production Research*, 26, 1119-1124.
- Hwang C L and A S M Masud (1979). Multiple Objective Decision Making- Methods and Applications: A state-of-the-art Survey. Springer-Verlag.
- Hwang C L, A S M Masud, S R Paidy and K Yoon (1982). Mathematical Programming with Multiple Objectives: A Tutorial. *Computers and Operations Research: A Special Issue on Mathematical Programming with Multiple Objective*, 7(1-2).
- Ignall E and L Schrage (1965). Application of the Branch and Bound Technique to some Flowshop Scheduling Problems, *Operations Research*, 13, 400-412.
- Ischibuchi H, N Yamamoto, T Murata, and H Tanaka. (1994). Genetic Algorithms and Neighbourhood Search Algorithms for Fuzzy Flowshop Scheduling Problems, *Fuzzy Sets and Systems*, 67, 81-100.
- Jain A K and H A Elmaraghy (1997). Production Scheduling/Rescheduling in Flexible Manufacturing, *Int'l J of Production Research*, 35, 281-309.
- Jain Nitin (1999). A Study of Hybridized Genetic Algorithms: An Application to Flowshops and Job Shops, *M Tech Thesis*, Department of Industrial and Management Engineering, Indian Institute of Technology Kanpur.
- Jayaram K (1998). Multi-objective Production Scheduling, *M Tech Thesis*, Department of Industrial and Management Engineering, Indian Institute of Technology Kanpur.
- Johnson E (1968). *Studies in Multi-objective Decision Models*, Monograph No. 1, Economic Research Centre, Lund, Sweden.
- Johnson S (1954). Optimal Two- and Three-Stage Production Schedules with Setup Times Included, *Naval Logistics Quaterly*, 1, 61-68.
- Keeney R (1983). *Decisions with Multiple Objectives-Preferences and Value Trade-offs*, John Wiley and Sons.
- Kennedy S (1993). Five Ways to a Smarter Genetic Algorithm, *AI Expert*, 35-38.
- Knuth D E (1983). *The Art of Computer Programming*, 3, Searching, Addison-Wesley.

- Kobayashi S, I Ono and M Yamamura. (1995). An Efficient Genetic Algorithm for Jobshop scheduling programs, in Eshelman (1995), 506-511.
- Krishnakumar K (1989). Micro-Genetic Algorithms for Stationary and Non-Stationary Function Optimization, SPIE Vol. 1196, *Intelligent Control and Adaptive Systems*.
- Kubota A (1995). Study on Optimal Scheduling for Manufacturing System by Genetic Algorithm, *Master's Thesis*, Ashikaga Institute of Technology, Ashikaga, Japan.
- Kumar Subodha (1997). An Application of Genetic Algorithms in No-wait Lot Streaming Flowshop Scheduling, *M Tech Thesis*, Department of Industrial and Management Engineering, Indian Institute of Technology Kanpur.
- Law Averill M and W David Kelton, (1991). *Simulation Modeling and Analysis*, 2nd edition, McGraw-Hill.
- Lenstra J, A Rinnooy Kan and P Brucker, (1977). Complexity of Machine Scheduling Problems, *Annals of Discrete Mathematics*, 1, 343-362.
- Lietmann G and A Marzollo (eds.) (1975). *Multicriteria Decision Making*, Springer-Verlag.
- Lin S C, E D Goodman and W F Punch (1997). Investigating Parallel Genetic Algorithms on Job Shop Scheduling Problems, Working Paper, GARA Group, Michigan State University.
- Magazine M J (1990). Notes on Scheduling and Sequencing, *Management Sciences*, University of Waterloo.
- Magill F N (1991). Magill's Survey of Science: Life Science Series, Vols. 1, 3 and 4, Salem Press, NJ.
- Manner R and B Manderick (ed.) (1992). *Parallel Problem Solving from Nature: PPSN II*, Elsevier Science Publishers, North-Holland
- McKay K, F Safayeni F and Buzacott, J (1988). Job-Shop Scheduling Theory: What is Relevant, *Interfaces*, 18-4, 84-90.
- Mendel George (1866). Experiments in Plant Hybridization, in *Classic Papers in Genetics*, J A Peters (ed.) (1959), Prentice-Hall.
- Metropolis N, A Rosenbluth, M Rosenbluth, A Teller, and E Teller (1953). Equation of State Calculations by Fast Computing Machines, *Journal of Chemical Physics*, 21, 1087-1092.
- Mitchell Melanie (1996). *An Introduction to Genetic Algorithms (Complex Adaptive Systems)*, MIT Press.
- Montgomery D C (1996). *Design and Analysis of Experiments*, 4th edition, John Wiley and Sons.
- Montgomery D C and George, C R (1998). *Applied Statistics and Probability for Engineers*, John Wiley and Sons.

- Morris G M, D S Goodsell, R S Halliday, R Huey, W E Hart, R K Belew and A J Olson (1998). Automated Docking using a Lamarckian Genetic Algorithm and Empirical Binding Free Energy Function, *J. Computational Chemistry*, 19, 1639-1662.
- Morton T and D Pentico (1993). *Heuristic Scheduling Systems--With Applications to a Production Systems and Project Management*, John Wiley & Sons.
- Muhlenbein Heinz (1997). Genetic Algorithms (137-171) in Aarts, Emile and J K Lenstra (1997), *Local Search in Combinatorial Optimization*, John Wiley & Sons.
- Murata T, H Ishibuchi (1994). Performance Evaluation of Genetic Algorithms for Flowshop Scheduling Problems, *IEEE*, 812-817.
- Murata T, H Ishibuchi and H Tanaka. (1996). Multiobjective Genetic Algorithms for Flowshop Scheduling Problems, *Computers and Industrial Engineering*, 30-4, 1061-1071.
- Muth J and G Thompson (ed.) (1963). *Industrial Scheduling*, Prentice Hall.
- Myers J and W H Carter Jr. (1973). Response Surface Techniques for Dual Response Systems, *Technometrics*, 16, 1973, 301-317.
- Nakano R and T Yamada (1992a). A Genetic Algorithm Applicable to Large Scale Jobshop Problems, in Manner and Manderick (1992), 281-290.
- Nakano, R and T Yamada (1992b). Conventional Genetic Algorithm for Jobshop Problems, in Belew and Booker (12), 477-479.
- Nawaz M, E Enscore and I Ham (1983). A Heuristic Algirithm for the m -Machine n -Job flowshop Sequencing Problem, *Omega*, 11, 11-95.
- Ogbu F and D Smith (1991). Simulated Annealing for the Flowshop Problem, *Omega*, 19, 64-67.
- Osyczka A and S Kundu (1996). A Modified Distance Method For Multicriteria Optimization, Using Genetic Algorithms, *Computers and Industrial Engineering.*, 30-4, 871-882.
- Palmer D (1965). Sequencing Jobs Through a Multi-stage Process in the Minimum Total Time—A Quick Method of Obtaining a Near Optimum, *Operations Research Quarterly*, 16, 101-107.
- Panwalkar S and W Iskander. (1977). A Survey of Scheduling Rules, *Operations research*, 25, 45-61.
- Paredis J (1992). Exploiting Constraints as Background Knowledge for Genetic Algorithms: a Case-Study for Scheduling, in Manner R and Manderick B (ed.) (1992), *Parallel Problem Solving in Nature*, 2, North-Holland, 281-290.

- Pareto Vilfredo (1906). *Manual di Economia Politica*, translated by A.S. Schwier, Manual of Political Economy, MacMillan, 1971.
- Phadke M S (1989). *Quality Engineering and Robust Design*, Prentice-Hall.
- Pinedo M (1995). *Scheduling Theory, Algorithms and Systems*, Prentice-Hall.
- Pinedo M and L Schrage (1982). Stochastic Shop Scheduling: A Survey in Dempster, et. al., 1821-196.
- Pinel J and K Singhal (1980). *Tolerance Analysis and Design*, Elsevier.
- Prasad Jugal (1997). A Study in Multi-Response Robust Design Using Genetic Algorithms, *M Tech Thesis*, Department of Industrial and Management Engineering, Indian Institute of Technology Kanpur.
- Rajendran C (1995). Heuristics For Scheduling in Flowshop With Multiple Objectives, *Eu. J. of Operational Research*, 82, 542-555.
- Raman Krishan (1994). An Evaluation of Genetic Algorithms for Robust Design, *M Tech Thesis*, Department of Industrial and Management Engineering, Indian Institute of Technology Kanpur.
- Rao P N (1999). Multi-objective Optimization of Injection Molding Process Parameters: A Study using Genetic Algorithms, *M Tech Thesis*, Department of Industrial and Management Engineering, Indian Institute of Technology Kanpur.
- Rao S S (1991). *Optimization Theory and Application*, New Delhi, Wiley Eastern Limited.
- Rawlings G (ed.) (1991). *Foundation of Genetic Algorithms*, Morgan Kaufmann.
- Reddi S S and C V Ramamoorthy (1972). On the Flowshop Sequencing Problem with No Wait in Process, *Operational Research Quarterly*, 23, 323-331.
- Reeves C (1995). A Genetic Algorithm for Flowshop Sequencing, *Computers and Operations Research*, 22, 5-13.
- Reklaitis G V, A Ravindran and K M Ragsdell (1983). Engineering Optimization: Methods and Application, John Wiley and Sons.
- Renders J and H Bersini (1994). Hybridizing Genetic Algorithms with Hill Climbing Methods for Global Optimization : Two Possible Ways, in Fogel (1994), 312-317.
- Rich Elaine and Kevin Knight (1983). *Artificial Intelligence*, McGraw-Hill.
- Richardson J T, M R Palmer, G Liepins and M Hillard (1989). Some Guidelines for Genetic Algorithms for Penalty Functions, in Schaffer et al. (1989), 191-197.
- Rinnooy Kan A (1976). *Machine Scheduling Problem: Classification, Complexity Computation*, Martinus Nijhoff, The Hague.

- Rintala T (1998). Parallel (diffusion) GAs, www.uwasa.fi/cs/publications.
- Ritzel B J, J W Eheart and S Ranjithan (1994). Using Genetic Algorithms to solve a Multiple Objective Ground Water Pollution Containment Problem, *Water Resources Research*, 30 (5), 1589-1603.
- Rosenberg R S (1967). Simulation of Genetic Populations with Biological Properties, *Ph D Dissertation*, University of Michigan.
- Roy B (1971). Problems and Methods with Multiple Objective Functions, *Mathematical Programming*, 1, 239-266.
- Russell Peter (1998). *Genetics*, 5th edition, Benjamin/Cummings.
- Sannomiya N and H Iima (1996). Application of GAs to Scheduling Problems in Manufacturing Processes, *Proceedings*, IEEE Conference on Evolutionary Computation, IEEE Press.
- Schaffer J D (1984). Some Experiments in Machine Learning using Vector Evaluated Genetic Algorithms (TCGA file No. 00314). *Ph D Dissertation*, Vanderbilt University.
- Schaffer J D (1985). Multiple Objective Optimization with Vector Evaluated Genetic Algorithms in Grefenstette J J (ed.) (1985), 93-100.
- Schaffer J D, R A Caruana, L J Eshelman and R Das (1989). A Study of Control Parameters Affecting Online Performance of Genetic Algorithms for Function Optimization, *Proceedings of the Third Int'l Conference on Genetic Algorithms*, Morgan Kauffmann, 51-60.
- Sen T and S Gupta (1984). A State-of-the-Art Survey of Static Scheduling Research involving Due Dates, *Omega*, 12, 63-76.
- Seo Fumiko and Masatoshi Sakawa (1988). *Multiple Criteria Decision Analysis in Regional Planning : Concepts, Methods and Applications*, Reidel, Boston.
- Shang J S and P R Tadikamalla (1998). Multicriteria Design and Control of a Cellular Manufacturing System through Simulation and Optimization, *Int. J. Prod. Research*, 36, 6, 1515-1528.
- Sharma Puneet (1996). Multicriteria Robust Design Using Non-dominated Sorting Genetic Algorithms, *M Tech Thesis*, Department of Industrial and Management Engineering, Indian Institute of Technology Kanpur.
- Simon H A (1969). *Science of the Artificial*, MIT Press.
- Smith Adam (1937). *The Wealth of Nations*, New York: Modern Library Inc.
- Smith John M (1993). *Theory of Evolution*, Canto, Cambridge University Press.

- Spears W M and K A De Jong, (1991). An Analysis of Multi-point Crossover in *Foundations of Genetic Algorithms*, G J E Rawlins (ed.), 310-315.
- Spencer Herbert (1910). *The First Principles of Heredity*, Blacks, London
- Spillman R (1993). Genetic Algorithms—Nature's Way to Search for the Best, *Dr. Dobb's Journal*, 26-30.
- Srinivas M and L M Patnaik (1994). Adaptive Probabilities of Crossover and Mutation in Genetic Algorithms, *IEEE Transactions on Systems, Man, and Cybernetics*, 24, 4, 656-667.
- Srinivas N and K Deb (1995). Multiobjective Optimization using Nondominated Sorting Genetic Algorithm, MIT Press, *Evolutionary Computations*, 2(3), 221-248
- Stahl Franklin W (1964). *The Mechanics of Inheritance*, Prentice-Hall.
- Starkweather T, S McDaniel, K Mathias, D Whitley and C Whitley (1991). An Comparison of Genetic Sequencing Operators, *Proceedings of Fourth Int'l Conference on Genetic Algorithms*, 69-76.
- Storer R, S Wu and R Vaccari (1992). New Search spaces for Sequencing Problems with Application to Jobshop Scheduling, *Management Science*, 38-10, 1495-1510.
- Suh N P (1990). *The Principles of Design*, Oxford University Press.
- Tabucanon M T (1989). *Multiple Criteria Decision Making in Industry*, Elsevier Science Publishers.
- Tackett W A (1994). Recombination, Selection and Genetic Construction of Computer Programs, *Ph D Thesis*, Dept. of Computer Engineering, University of Southern California.
- Taguchi G (1986). *Introduction to Quality Engineering*, Asian Productivity Organization, Tokyo.
- Taillard E (1993). Benchmarks for Basic Scheduling problems, *Eur. J. Operational Research*, 64, 278-285.
- Tamaki H and Y Nishikawa (1992) a Parallel Genetic Algorithm based on a Neighbourhood Model and its Applications to the Jobshop Scheduling, in Manner and Manderick (1992), 573-582.
- Tamura K and S Miura (1979). Necessary and Sufficient Conditions for Local and Global Nondominated Solutions in Decision Problems with Multiobjectives, *Journal of Optimization Theory and Applications*, 27, 509-523.
- Uckun S, S Bagchi, K Kawamura and Y Miyabe (1993). Managing Genetic Search in Job Shop Scheduling, *IEEE Expert*, October.
- Van Wassenhove L and L F Gelders (1980). Solving a Bicriterion Scheduling Problem, *Eur. J. of Operational Research*, 4, 42-48.

- Van Wassenhove L and K R Baker (1982). A Bicriterion Approach to Time/Cost Trade-offs in Sequencing, *Eur. J. Operational Research*, 11, 48-54.
- Vining G G and R H Myers (1990). Combining Taguchi and Response Surface Philosophies: A Dual Response Approach, *J. Quality Technology*, 22, 38-45.
- Vose M D (1993). Modeling Simple GAs in LD Whitley (ed.), *Foundations of Genetic Algorithms 2*, Morgan Kaufmann.
- Vose M D and Liepins, G E (1991). Punctuated Equilibria in Genetic Search, *Complexity*, 5, 31-44.
- Watson J D (1968). *The Double Helix*, Atheneum.
- Watson J D and Crick, F H C (1953). Genetical Implications of the Structure of Deoxyribonucleic Acids. A Structure for DNA, *Nature*, 171: 737-738.
- Weiner Jonathan (1995). *The Beak of the Finch: A Story of Evolution in Our Time*, Vintage Books.
- Wellman and Gemmill (1995). A Genetic Algorithm Approach to Optimization of Asynchronous Automatic Assembly Systems, *Int'l J. Flexible Manufacturing Systems*, 7, 27-46.
- Whitley L D (1989). The GENITOR Algorithm and Selection Pressure: Why Rank Based Allocation of Reproductive Trials is Best, *Proceedings of Third Int'l Conference on Genetic Algorithms*, Morgan Kauffman.
- Whitley L D and M D Vose (1995). *Foundations of GAs 3*, Morgan Kaufmann.
- Wilmer P (1990). *Invertebrate Relationships: Patterns in Animal Evolution*, Cambridge University Press.
- Young J Z (1971). *An Introduction to the Study of Man*, Oxford.
- Zegrodi S H, K Itoh and T Enkawa (1995). Minimizing Makespan for Flowshop Scheduling by Combining Simulated Annealing with Sequencing Knowledge, *Eur. J. Operational Research*, 85, 515-531.
- Zeleny Milan (1985). *Multiple Criteria Decision Making*, McGraw-Hill.
- Zweben Monte and Mark S Fox (1994). *Intelligent Scheduling*, MorganKaufmann.

Appendix

C++ Codes for a Hybridized GA to Sequence the Single-Objective Flowshop

Henry Ford achieved dramatic cost savings after introducing the moving belt assembly line into automobile manufacturing in 1913. Ever since then it has been believed that the flow line is perhaps the ideal way to organize manufacturing for products made in sufficient volume to justify the investment in dedicated work stations and a dedicated material handling system (Buzacott and Shanthikumar, 1992). Since then, the flow line has been widely used for assembly operations in automobile, domestic appliance, and high volume electronic products industries.

The flow line produces identical objects, each requiring exactly the same operations—always done in the same sequence. The challenge in managing the flow line is to balance the work at the different workstations, to minimize idling at the stations. The flowshop, on the other hand, has more flexibility. In it, the jobs are not identical, but they require the *same sequence* of operations to be performed on them in the same sequence. Thus, all jobs flow here from station to station along a single, identical path. Clearly, since the amount of processing required on each job at a given station might be different, idle times may develop at the stations (Figure 1.1). The challenge in a flowshop is to find the *optimum sequence* in which the jobs must be fed to the first station such that such idle times are minimized (Section 4.1).

Chapter 4 reviewed the methods for optimally sequencing jobs in a flowshop. This Appendix contains a code that solves the flowshop by the better-performing heuristic methods described in Chapter 4, as well as by Darwinian and Lamarckian (hybridized) genetic algorithms. The output is tabular, stored in an output file for subsequent review and analysis, as well as graphical (see page 340).

README . TXT

Software copyright Nitin Jain and T P Bagchi 1999

**Disclaimer: User assumes full risk of all consequences
----- of using this software.**

WHAT DOES THIS SOFTWARE DO?

This software is designed to sequence jobs in a static FLOWSHOP. The software uses "GENETIC ALGORITHMS" (Darwinian and Darwinian-Lamarckian) as well as conventional heuristics to tackle this NP-hard combinatorial problem. Solutions obtained are expected to be near optimal.

HOW TO USE THIS SOFTWARE

Solution method has been coded in Boreland Turbo C++® and put in 9 files.

To produce an executable file out of these program files, invoke Boreland Turbo C++ and ADD all *.cpp files (but DO NOT INCLUDE chrom.h) into one C++ project. Assign a name of your choice to your project when prompted by Turbo C++.

Program Files

MAIN.CPP { Contains the main sklenton of GA processing}
OPER.CPP { Have the modules of genetic operators}
CHROM.CPP { Defines the chromosome and Job classes}
RANDOM.CPP { Defines the class of random numbers}
HEURISTI.CPP { Contains solution-generating heuristics}
FHO.CPP { Module for HO_CHANG Heuristic}
BAGCHI.CPP {Contains all cosmetics--input/output etc.}
GRAPH.CPP {Has the module for plotting the final graph}

Note 1: The following file defines all object classes used in the program. It does not have to be included in the PROJECT, but it must exist in the directory containing the executable files of the PROJECT

CHROM.H { Header file; Declares the classes}

Note 2: TURBO C++ SOURCE DIRECTORY contains all *.CPP AND *.h FILES. The OUTPUT DIRECTORY contains *.OBJ,

problem data file (e.g. in.ga) and the resutls.dat file. It also contains all *.EXE files produced after compilation.

INPUTS REQUIRED

A separate Data File must contain your Problem Data (Processing times, etc.). When executed, the program will ask for a data file that contains the details of flowshop problem, i.e. no. of machines, no. of jobs and processing times of different jobs on machines. The format to be used in this file is as follows:

Let the problem be of sequencing 4 jobs on 3 machines and let p_{ij} represent the processing time of the i th job on m th machine. Then the data will be as

```
4 3
p11 p12 p13
p21 p22 p23
p31 p32 p33
p41 p42 p43
```

Other than the above problem data, the program will require you to enter the values of GA parameters, namely, population size and the probabilities of crossover and mutation.

The optimum values of these parameters are problem-specific, but the default parameter values already preset in program may not be a bad choice...

Finally, you must specify (a) the GA termination criterion i.e. no. of generations the GA should process up to, and (b) a random "Seed no.". The termination criterion depends upon the size of problem.

For a large problem GA should be allowed to run for longer periods (generations) and vice-versa. But, there is no harm in allowing the GA to run a bit longer if computer time is not at premium.

Each different random seed generates a different stream of random numbers and thus controls the starting point and by some extent the direction of GA search. With different seeds GA search may end up at different solutions. But a good choice of GA parameters

(population size, probability of crossover and the probability of mutation) can make the GA robust. Such robustness would reduce the sensitivity of the GA to the initial population.

OUTPUT

The output of the program will be displayed on monitor screen as well as stored in one file named "**results.dat**". The Results.dat file will contain the values of the best, worst and average makespans found in the whole population at each generation, starting from generation zero (the initial population) to the last generation executed.

At the end of file the best job sequence found by GA processing will be displayed.

The graphical plot displays the convergence experienced by the GA. The plot on the screen may be printed by pressing "shift+Print Screen keys", provided you have run the utility

GRAPHICS.COM LASERJETII

at the DOS prompt before you attempt to print the screen.

Any other screen may be printed on the printer at any time by pressing Shift+Print Screen keys.

Welcome to the world of GAs!

```

//      File 1
//      MAIN.CPP
//      For details, please go through the file 'README.TXT'
//      Copyright Nitin Jain and T P Bagchi, February 26, 1999
//      Declaration of Standard Functions of Borland Turbo C++( V. 3 Library

#include <conio.h>
#include <stdlib.h>
#include <fstream.h>
#include <graphics.h>
#include <math.h>
#include "chrom.h"
#include "string.h"
#include <dos.h>

//      PROGRAM VARIABLES

int POP=100,GEN=100;
int NMC=15,SEED=1;
int NJOBS=75;
chrom * chromosom=new chrom[POP+1]; ;
Job *job= new Job[NJOBS+1];
class random ran;
int preserve=1,MULT=0;
float CROSS_PROB=.7, MUT_PROB=.01,doe[10];
ifstream fin;
ofstream fout, fout1,foutx;
chrom bestchrom, worst, H_opt,ref;
char option1=0,option2=0,option3='n';
char *file;
extern float **dr;
extern int R,P,N,C;
chrom SBEST;
float Sworst,Savg,spread;

//      FUNCTIONS used in the program

void INITIALIZATION();           //Initialize the ga
void REPRODUCTION(void);        //genetic operator { in oper.cpp}
void CROSSOVER(void);           //genetic operator {      "      }
void MUTATION(void);            //genetic operator {      "      }
void box(int x1, int y1,int x2,int y2,int bkc,int textc,int shadow);
void Preserve_the_best(int);    // implement elitism strategy
void Sort(float *, int *, int);
void Collect_Data(int );         // to collect statistics
void Evaluate(void);             // to evaluate the whole population
void Kick_Worst(void);          // for elitism strategy
void REPORT(int );              // dispaly statistics
void FREPORT(void);             // dispaly final statistics
void PALMER(chrom &);          // defined in heuristi.cpp
void NEH(chrom &);             //
void RA(chrom &);              //
void CDS(chrom &);             //
void I_HO(void);               // initilizes the HO_CHANG heuristic { in fho.cpp}

void HEURST_HO(chrom &);
void IF_NEEDS_APPLY_HO( chrom &);
void Menu(int a=0);
void graph();                   // display graph
void kill();
void killho();                  // memory management
void S_files(void);             // Start output files
void SHOW_T();

main(){
    int i,k,j,gen=0;
    int x=0, mcount;
    bestchrom.string=new int[NJOBS];
    ref.string=new int[NJOBS];
    H_opt.string=new int[NJOBS];
}

```

```

        worst.string=new int[NJOBS];
        SBEST.string=new int[NJOBS];
        SBEST.cmax=10e10;
        Menu();
        foutx.open("results.dat");
X:      if(x!=0) { Menu(1); gen=0; }
        if(option1=='3'||option1=='4') {MULT=1;SEED=1;mcount=1;
        SBEST.cmax=10e10; Sworst=0;Savg=0; }
        else{
            fout.open("data1");
            fout1.open("data2");} S_files();
        do{
            INITIALIZATION( );
            Evaluate();
            Preserve_the_best(preserve);
            if(MULT==0) Collect_Data(gen);

//      GA LOOP STARTS HERE.....
            while(GEN>gen) {
                cal_sprob(chromosom);
                cal_cumprob(chromosom);
                REPRODUCTION();
                CROSSOVER();
                MUTATION();
                Evaluate();

                Kick_Worst();
                Preserve_the_best(preserve);
                if (option3=='Y'|| option3=='y') IF_NEEDS_APPLY_HO(chromosom[POP]);
                if(MULT==0) Collect_Data(gen+1);

                gen++;
            }
            if(MULT==1){ if (mcount<10) delete[] chromosom;
            doe[mcount-1]=bestchrom.cmax;
            SEED++; mcount++;
            if (option3=='y'||option3=='Y') killho(); gen=0;
            textbackground(2);gotoxy(21+mcount*3,17);cputs("    ");
            if (SBEST.cmax>bestchrom.cmax) SBEST=bestchrom;
            foutx<<SEED-1<<"<<bestchrom.cmax<<\n";
            if(Sworst>bestchrom.cmax)Sworst=bestchrom.cmax;
            Savg+=bestchrom.cmax/10.0;
            }
            } while(MULT!=0 && mcount<11); // end do
            x=1;
            if(mcount==11){
                spread=Sworst-SBEST.cmax;
                foutx<<"\nBEST MAKESPAN = "<<SBEST.cmax;
                foutx<<"\nAVERAGE MAKESPAN= "<<(int)Savg<<"\nSpread = "<<spread<<"\n\n";
                MULT=0;mcount=1;SHOW_T ();
                window(16,7,63,24);
                textcolor(6);gotoxy(1,15);textcolor(13);cputs("BEST SEQUENCE:");
                gotoxy(1, 16); textcolor(12);SBEST.show();
                window(1,1,80,25); getch();
                goto X;}
                FREPORT();
            goto X;
        }

//      ... GA LOOP ENDS HERE

//      Routine initializes random number generator and population

void INITIALIZATION( ){
    int i;
    bestchrom.cmax=10e10;
    worst.cmax=0;
    srand(SEED); // start the random generator
if( option3=='Y'|| option3=='y') I_HO();
    if( option1=='2'||option1=='4'){
        for(i=0;i<NJOBS;i++) H_opt.string[i]=i+1 ;
}

```

```

        switch( option2) {
            case '1' :      PALMER(H_opt);
                break;
        case '2' :      NEH(H_opt);
                break;
            case '3' :      CDS(H_opt);
                break;
            case '4' :      RA(H_opt);
                break;
        default   : { cout<<" Buggggggggggggggggggg!!!!!!";
                        exit(1); }
                } // closes switch
        }

chromosom= new chrom [POP+preserve]; //allocates memory to chromosom array

{ Evaluate(); REPORT(-1);           // for graph only
  if ( option1=='2'||option1=='4') for(i=0;i<POP;i++)
    chromosom[i]=H_opt;
} // close initialization

void Evaluate(void){
int i;
  for (i=0; i<POP;i++) chromosom[i].cal_stat();
}
void Collect_Data(int gen) { // Collects data after every generation
REPORT(gen);
}

// MODULES FOR ELITISM STRATEGY

void Preserve_the_best(int preserve){
int i,j,* index= new int [POP+preserve];
float * x= new float [POP+preserve];
for(i=0;i<POP+preserve;i++)
  x[i]=chromosom[i].cmax;
index[i]=i;
}
Sort(x,index,POP);
for(i=0;i<preserve;i++)
if (bestchrom>chromosom[index[i]])
chromosom[POP+preserve-i-1]=chromosom[index[i]];
bestchrom=chromosom[POP+preserve-1];
delete x;
delete index;
}

// Routine "bubble sorts" x
// Sorting is used in finding the best solution, worst solution, etc.

void Sort(float *x,int *index, int a){
int i=0,temp,i;
float temp;
for(i=0;i<a-1;i++){
  for(j=i+1;j<a;j++)
    if (x[i]>x[j]){
      temp=x[j];tempi=index[j];
      x[j]=x[i];index[j]=index[i];
      x[i]=temp;index[i]=tempi;
    }
}
}

void Kick_Worst(void){
int i,j,* index= new int [POP+preserve];
float * x= new float [POP+preserve];
for(i=0;i<POP;i++){
  x[i]=chromosom[i].cmax;
  index[i]=i;
}
Sort(x,index,POP);
}
```

```

if (option3!='Y' && option3=='y')  {
if( ran.zero_one(.12) )
for(i=0;i<preserve;i++){
if (chromosom[index[POP-i-1]]>chromosom[POP+preserve-i-1])
chromosom[index[POP-1]]=chromosom[POP+preserve-i-1];
}
}
else{
ran.zero_one(1);
for(i=0;i<preserve;i++){
if (chromosom[index[POP-i-1]]>chromosom[POP+preserve-i-1])
chromosom[index[POP-1]]=chromosom[POP+preserve-i-1];
}
}
delete x;
delete index;
}
void IF_NEEDS_APPLY_HO(chrom & o){
static int count=0;
    if (bestchrom>o)
        count=0;
        count++;
    if(count==6) {
        HEURST_HO(o) ;
    }
}

// MODULES FOR USER INPUT DATA COLLECTION AND DISPLAY

void S_files() {
if (MULT==1)  foutx<<"\n RESULTS WITH MUTIPLE SEEDS\n";
if( option1=='2'){
switch( option2) {
case '1' : foutx<<" INITIALIZE BY PALMER HEURISTIC\n "; break;
case '2' : foutx<<" INITIALIZE BY NEH HEURISTIC\n "; break;
case '3' : foutx<<" INITIALIZE BY CDS HEURISTIC\n "; break;
case '4' : foutx<<" INITIALIZE BY RA HEURISTIC\n "; break;
}
}
if( option3=='Y' || option3=='y')  foutx<<" AND HO_CHAG ASSISTED\n\n";
else foutx<<"Standard Ga\n";
foutx<<"POP<<"\n"<<CROSS_PROB<< "\n"<<MUT_PROB<<"\n"<<GEN<<"\n";
foutx<<"POP=<<POP<<"\n"<<"Pm=" <<CROSS_PROB;
foutx<<"\n"<<"Pm=<<MUT_PROB<<"\n"<<"Gen= "<<GEN<<"\n";
if(!MULT) {
foutx<<"\n SEED #=" <<SEED;
foutx<<"          Best Makespan      Worst Makespan      Avg.Makespan\n";
else foutx<<" \n SEED#      BEST MAKESPAN FOUND\n";
}
}

void REPORT(int gen){
if( gen!=-1){
char static str[8]={ ' ', ' ', ' ', ' ', ' ', '\0', ' ', ' ',};

foutx<<" Gen # "<<gen<<"           ";
window(16,5,63,20);
gotoxy(32 ,6); cout<<str;
gotoxy( 32,6);
cout<<bestchrom.cmax;
gotoxy(32 ,4); cout<<str;
gotoxy( 32,4); cout<<gen;
int i; float sum=0; worst.cmax=0;
for ( i=0;i<POP; i++){
if (worst.cmax<chromosom[i].cmax)
worst=chromosom[i];
sum+=float(chromosom[i].cmax);
}
int bb = (sum/POP);
fout1<<gen<<" ";
if( gen==0)
fout<<" <<bestchrom.cmax<<"     "<<worst.cmax<<" << bb<<"\n";
fout1<<"     "<<bestchrom.cmax<<"     "<<worst.cmax<<" <<bb<<"\n";
foutx<<"     "<<bestchrom.cmax<<"     "<<worst.cmax<<" <<bb<<"\n";
}

```

```

gotoxy(32 ,8); cout<<str;
gotoxy( 32,8); cout<<bb;
gotoxy(32 ,10); cout<<str;
gotoxy( 32,10); cout<<worst.cmax;
gotoxy(1,13); cputs("Best Sequence : "); textcolor(6);
gotoxy(1, 15); bestchrom.show(); textcolor(14);

window(1,1,80,25);
} else {ofstream f0; f0.open("x");
int i; float sum=0; worst.cmax=0;
for ( i=0;i<POP; i++) {
if (worst.cmax<chromosom[i].cmax)
worst=chromosom[i];
sum+=float(chromosom[i].cmax);
}
int bb = (sum/POP);
Preserve_the_best(1) ;

f0<<"<<bestchrom.cmax<<" <<worst.cmax<<"<<bb<<"\n";
f0.close(); }
}

// Display results, final statistics and graph...

void FREPORT(void){
gotoxy( 32,22);
cout<< "END OF GA RUN.;" gotoxy(21,24);
fout<<"\n"<<bestchrom.cmax ;
cputs(" Press Q to quit.Any other key to continue"); sleep(1);
gotoxy(29,22); cputs("Press G for convergence graph");
fout.close(); fout1.close();
foutx<<"\n\n BEST SEQUENCE FOUND\n\n";
{ int i;
for (i=0; i<NJOBS;i++){
foutx<<bestchrom.string[i];foutx<<".";
}
foutx<<"\n";
}

{ char x; x= getch(); if( x=='q' || x=='Q') { system("cls");exit(1);}
if( x=='g' || x=='G') graph();
system("cls"); gotoxy(20,10);
cputs(" Press Q to quit.Any other key to continue";
{ char x; x= getch(); if( x=='q' || x=='Q') {
system("cls");exit(1);}
}

}

// Memory management

void kill(){
delete [] chromosom;
delete [] job; }
void killho(){
for(int i=0;i<NJOBS;i++)
delete dr[i];
delete dr;
}

// Display GA results for multiple seeds...

void SHOW_T(){
system("cls");
box(2,2,79,25,15,15,0);
box(10,4,73,24,9,15,0);textcolor(12);

gotoxy(12,6); cputs("PALMER");gotoxy(13,8); cout<<P;
gotoxy(22,6); cputs("NEH"); gotoxy(22,8); cout<<N;
gotoxy(29,6);cputs("CDS"); gotoxy(29,8); cout<<C;
gotoxy(36,6);cputs("RA"); gotoxy(36,8); cout<<R; textcolor(13);
gotoxy(42,6);cputs("SEED#"); textcolor(12);

```

```

if(option1=='3'){
    gotoxy(48,6); cputs("BEST MAKESPAN BY SGA");}else
{gotoxy(48,6); cputs("BEST MAKESPAN BY HGA");}
for(int i=0;i<10;i++)
{ gotoxy(55,i+8); cout<<doe[i]; gotoxy(44,i+8);
cout<<i+1;}
gotoxy(40,18);cputs("Best Makesapn      : ");cout<<SBEST.cmax;
gotoxy(40,19);cputs("Average Makesapn : ");cout<<(int)Savg;
gotoxy(40,20);cputs("Spread           : ");cout<<(int)spread;

    gotoxy(48,7);
        if( option1=='2'||option1=='4'){
            switch( option2 ) {
                case '1' : cputs(" PALMER Initiated"); break;
                case '2' : cputs(" NEH Initiated"); break;
                case '3' : cputs(" CDS Initiated"); break;
                case '4' : cputs(" RA Initiated"); break;
            }
        }
        if (option3=='y'||option3=='Y')cputs("& Ho Aided");
}

//      File 2  GENETIC OPERATORS
//      OPER.CPP

#include<fstream.h>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "chrom.h"
extern int POP, NJOBS, NMC, preserve;
extern class random ran;
extern class Job *job;
extern float MUT_PROB, CROSS_PROB;
extern chrom *chromosom;

void REPRODUCTION(void){          // fuction for reproduction

    int ** duplicate;
    int i,j,k;
    duplicate= new int * [POP];
    for( i=0; i<POP;i++) duplicate[i] = new int [NJOBS];
    for (i=0; i<POP;i++){
        for( j=0;j<NJOBS;j++)
            duplicate[i][j]=chromosom[i].string[j];
    }

    for(i=0; i<POP;i++){
        k=ran.r_wheel(chromosom);
        for(j=0;j<NJOBS;j++)
            chromosom[i].string[j]=duplicate[k][j];
    }

    for( i=0;i<POP;i++) delete duplicate[i] ;
    delete duplicate;
}

//      Code for Crossover Operator: One-Point Crossover

void CROSSOVER(void){
    int i,j,k,temp;
    int njobs=NJOBS;
    int pop,t1=0,t2=0;
    int * templ= new int [NJOBS];
    int * temp2= new int [NJOBS];

// we need even no. of chromosomes for crossover ...
    if (POP%2!=0)pop=POP-1;
    else pop=POP;
}

```

```

        for (i=0; i<pop;i=i+2){ // crossover loop

            // generate the bit position k in a chromosome
            k=ran.crossover(NJOBS,CROSS_PROB);

            if (k!=0){ // k=0 means, no crossover

                for (j=0;j<njobs;j++){
                    temp1[j]=chromosom[i].string[j];
                    temp2[j]=chromosom[i+1].string[j];
                }
                for (j=0;j<k;j++){
                    while(temp2[t1]!=chromosom[i].string[j]) t1++;
                    temp2[t1]=0;
                    while(temp1[t2]!=chromosom[i+1].string[j]) t2++;
                    temp1[t2]=0;
                    t1=0;
                    t2=0;
                }

                for(j=k;j<njobs;j++){
                    while(temp2[t1]==0) t1++;
                    chromosom[i].string[j]=temp2[t1];
                    temp2[t1]=0;
                    t1=0;
                    while(temp1[t2]==0) t2++;
                    chromosom[i+1].string[j]=temp1[t2];
                    temp1[t2]=0;
                    t2=0;
                }
            }
            delete temp1;
            delete temp2;
        }

//      Code for Mutation: Random swap of jobs mutation ...

void MUTATION(void){

    // loop for mutation
    int i,j,k,temp;
    int njobs=NJOBS;
    for (i=0; i<POP;i++) {
        for (j=0; j<NJOBS;j++) {
            if (ran.zero_one(MUT_PROB)==1) {
                k=ran.r_wheel(njobs);
                temp=chromosom[i].string[j];
                chromosom[i].string[j]=chromosom[i].string[k];

                chromosom[i].string[k]=temp;
            }
        }
    }
}

//      File 3  CLASS OF CHROMOSOMES
//      CHROM.CPP
//      Code declares Classes (OOPS) for chromosomes and jobs

#include<fstream.h>
#include <stdlib.h>
#include <math.h>
# include "chrom.h"
#include <conio.h>

extern int POP, NJOBS, NMC, preserve;
extern class random ran;
extern class ifstream fin;

```

```

extern class Job *job;
int chrom::njobs=0;           //initialization of static variable
int chrom::nmc=0;
int chrom::pop=0;
int chrom::count=0;
class random chrom::ra;

chrom:: chrom(){
// constructor: produces a feasible chromosome (solution) randomly
    nmc=NMC;
    njobs=NJOBS;
    pop=POP;
    string= new int [njobs];
    schedule(njobs);
}

void chrom::operator=(chrom& a){
    int i;
    for(i=0;i<njobs;i++)
        string[i]=a.string[i];
    cmax=a.cmax;
}

int chrom:: operator>(chrom & a){
    if ( cmax<=a.cmax) return(0);
    else return(1);
}

void chrom::schedule(int njobs){
    int *list;
    int i,j,k;
    list= new int[njobs];
    for (i=0;i<njobs;i++){
        list[i]=i+1;
    }
    for (i=0;i<njobs;i++){
        k=ran.r_wheel(njobs-i);
        string[i]=list[k];
        for(j=k;j<(njobs-1);j++)
            list[j]=list[j+1];
    }
    delete list;
}

void chrom:: cal_stat(int njobs){
    int i,j,fj;
    fj=string[0];
    job[fj].starting[0]=0;
    for(i=0;i<nmc;i++){

        job[fj].finishing[i]=job[fj].starting[i]+job[fj].ptime[i];
        if((i+1)<nmc)
            job[fj].starting[i+1]=job[fj].finishing[i];
    }
    for(i=1;i<njobs;i++){
        job[string[i]].finishing[0]=job[string[i-1]].finishing[0]+
        job[string[i]].ptime[0];
        job[string[i]].starting[0]=job[string[i-1]].finishing[0];
    }
    for(i=1;i<njobs;i++){
        for(j=1;j<nmc;j++){
            job[string[i]].starting[j]=max(job[string[i]].finishing[j-1],
            job[string[i-1]].finishing[j]);
            job[string[i]].finishing[j]=job[string[i]].starting[j]+
            job[string[i]].ptime[j];
        }
    }
    cmax=job[string[njobs-1]].finishing[nmc-1];
    count++;
}

float chrom::max(float a, float b){

}

```

```

float k;
k=(a>b)?a:b;
return(k);
}

chrom::~chrom(){
    delete string;
}

// calculate probability of reproduction...

void cal_sprob(chrom a[],int p){

    float popmax,sum=0;
    int i; p=0;
    for( i=0 ; i<a[0].pop+p ; i++)
    popmax=(a[i].cmax>popmax)?a[i].cmax:popmax;
    for ( i=0; i<a[0].pop;i++) {
        sum+=(popmax-a[i].cmax)*(popmax-a[i].cmax);
    }
    if (sum>.01){
        for ( i=0; i<a[0].pop;i++)
        a[i].s_prob=(popmax-a[i].cmax)*(popmax-a[i].cmax)/sum;
    } else
        for(i=0;i<a[0].pop+p;i++)a[i].s_prob=1.0/a[0].pop;
}

void cal_cumprob(chrom a[],int p){
    int i;
    a[0].cum_prob=a[0].s_prob;
    for ( i=1; i<a[0].pop+p;i++) {
        a[i].cum_prob=a[i-1].cum_prob+a[i].s_prob;
    }
    a[POP-1].cum_prob=1;
}

// Display the statistics (job sequence and makespan) for the chromosome..

void chrom:: show(int njobs){
    int i;
    for (i=0; i<njobs;i++){
        char str[10];
        itoa(string[i], str, 10);
        cputs(str);cputs(".");
    }
}

//      Reads processing times ..

void Job::readptime(){
int i, x;
for (i=0;i<nmc;i++)
    fin>>ptime[i];
}

Job::Job(){
    nmc=NMC;// cout<<" jobb";
    starting=new float[nmc];
    finishing=new float[nmc];
    ptime=new float[nmc];
}

Job::~Job(){
    delete starting;
    delete finishing;
    delete ptime; // cout<<"job killed ";
}

//      Routine to calculate start and finish times for each job ...

void show_stat(Job * job){
    int i,j;

```

```

        cout<<"\n\n\n";
        for(i=1;i<=NJOBS;i++) {
cout<<"-----";
cout<<" \n  JOB["<<i<<"] :          \n ";
cout<<"           S. time      F. time\n";
        for (j=0;j<NMC;j++)
cout<<"   M/C["<<j+1<<"] :          "<<job[i].starting[j]<<
"<<job[i].finishing[j]<<"\n";
}
}

//      File 4      CLASS OF RANDOM VARIABLES
//      RANDOM.CPP

#include <iostream.h>
#include <stdio.h>
#include <stdlib.h>

#include "chrom.h"

float random:: r_var(void){           // returns a random value between 0 and 1
    float x=rand();
    x=x/32767;
    if ( x>1 ) cout<< "ERROR IN RANDOM NO. GENERATION"; exit(1);
    return(x);
}

random::random(){                   //constructor
}

int random:: zero_one(float prob){

// return either 0 or 1, 1 with a prob of p and 0 with (1-p) ...

    if ((r_var())<= prob) return(1);
    else return(0);
}

int random:: crossover(int length, float prob){
//Note: length is the length of cromosome; prob is crossover probability

    float *cum =new float [length];
    float jj;
    int i;

if (( r_var())<=prob){
//      checks whether crossover will occur or not
    cum[0]=(float)1/length;
//  Decides bit position at which the 1-point crossover will occur...
    for ( i=1;i<length;i++)
        cum[i]=cum[i-1]+cum[0];
// each bit will have equal chance for being chosen as crossover pt.)
    cum[length-1]=1; // avoids rounding error problem
    i=0;

    float j=r_var();
    while (j>cum[i++]); // uniform probability roulette wheel
    delete cum;
    return(--i);
}
else { delete cum;
    return(0); // if crossover is not to take place
}
}

```

```

// function simulates a Roulette Wheel; g is the array of chromosome object

int random::r_wheel(chrom g[]){
    int i=0;
    float j;
    j=r_var();
    while (j>g[i++].cum_prob);
    return(--i);
}

int random::r_wheel(int a){
float j,k,*g;
int i;
g=new float[a];
k=1.0/a;
for (i=0;i<a;i++)
g[i]=k*(i+1);
g[a-1]=1; //to avoid any rounding error problem
i=0;
j=r_var();
while(j>g[i++]);
delete g;
return(--i);
}

//      File 5 Contains flowshop sequence solution-generating heuristics...
// HEURISTI.CPP

#include<fstream.h>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "chrom.h"
extern int POP, NJOBS, NMC ;
extern class random ran;
extern class Job *job;
void Sort(float *, int *, int);
void Johnson(float ** ,int * );

//      The CDS Heuristic ...

void CDS(chrom & a){
    float **VMC;
    int *order = new int[NJOBS];
    chrom best;   best.cmax=10e10;
    int i,j,k;
    void Clear(float **,int *);
    VMC=new float * [2];
    for(i=0;i<=1;i++) VMC[i]=new float [NJOBS+1];
    for(i=0;i<NMC-1;i++){
        Clear(VMC,order);
        for(j=1;j<=NJOBS;j++){
            for(k=0;k<=i;k++){
                VMC[0][j]+=job[j].ptime[k];
                VMC[1][j]+=job[j].ptime[NMC-1-k];
            }
        }
        Johnson(VMC,order);
        for(j=0;j<NJOBS;j++)
            a.string[j]=order[j];
        a.cal_stat();
        if(best>a) best=a;
    }
    a=best;
    for(i=0;i<=1;i++) delete VMC[i];
    delete VMC ;
    delete order;
}
void Clear(float ** VMC,int * order){


```

```

int i,j;
    for(i=0;i<2;i++) for (j=1;j<=NJOBS;j++)
        VMC[i][j]=0;
    for(j=0;j<NJOBS;j++) order[j]=j+1;
}

//      The NEH Heuristic ...

void NEH(chrom & o){

int nfix,* Fix=new int[NJOBS],* others= new int[NJOBS-1];
int i,j,k,t,* index= new int[NJOBS], testp;
float *Sum= new float[NJOBS], min=10e10;
chrom bestsofar;
    for(i=0;i<NJOBS;i++) Sum[i]=0;
    for(i=0;i<NJOBS;i++){

        for(j=0;j<NMC;j++) Sum[i]+=job[i+1].ptime[j];
        index[i]=i+1;
    }
Sort(Sum,index,NJOBS);
nfix=1; testp=nfix+1;
Fix[0]=index[NJOBS-1];

for(i=0;i<NJOBS-1;i++) others[i]=index[NJOBS-2-i];

for(i=0;i<NJOBS-1;i++){
    for(j=0;j<testp;j++){
        o.string[j]=others[i];
        t=0;
        for(k=0;k<nfix+1;k++) if(k!=j) o.string[k]=Fix[t++];
        o.cal_stat(nfix+1);
        if(min>o.cmax){
            min=o.cmax;
            bestsofar=o;
        }
        min=10e10;
        for(j=0;j<=nfix;j++)
            Fix[j]=bestsofar.string[j];
        nfix++;
        testp=nfix+1;
    }
    o=bestsofar;
    delete Fix;
    delete others;
    delete index;
    delete Sum;
}

//      The Palmer Heuristic ...

void PALMER(chrom & a){

float *order=new float [NJOBS], *S=new float[NJOBS+1];
int i,j,*index= new int[NJOBS];
    for(i=1;i<NJOBS;i++){ S[i]=0;
    for (j=0;j<NMC;j++)
        S[i]+=(2*(j+1)-NMC -1)*job[i].ptime[j];
    }
    for(i=0;i<NJOBS;i++){
        order[i]=a.string[i];
        index[i]=a.string[i];
    }
Sort(order,index,NJOBS);
    for(i=0;i<NJOBS;i++){
        a.string[i]=index[NJOBS-i-1];
    }
    delete index;
    delete order;
    delete S;
}

```

```

//      The Dannenbring RA Heuristic ...

void RA(chrom & a){
    float **VMC;
    int * order= new int [NJOBS],** W;
    int i,j,k;
    void Johnson(float **,int *);

    VMC=new float * [2];
    W=new int * [2];
    for(i=0;i<1;i++) VMC[i]=new float [NJOBS+1];
    for(i=0;i<1;i++) W[i]=new int [NMC];

    for(i=0;i<NMC;i++){
        W[0][i]=NMC-i;
        W[1][i]=i+1;
    }
    for(i=1;i<=NJOBS;i++){
        VMC[0][i]=0;
        VMC[1][i]=0;
    }

    for(i=1;i<=NJOBS;i++)
        for(j=0;j<NMC;j++){
            VMC[0][i]+=W[0][j]*job[i].ptime[j];
            VMC[1][i]+=W[1][j]*job[i].ptime[j];
        }
    for(i=0;i<NJOBS;i++) order[i]=i+1;
    Johnson(VMC,order);
    for(j=0;j<NJOBS;j++)
        a.string[j]=order[j];
    for(i=0;i<=1;i++) delete VMC[i];
    for(i=0;i<=1;i++) delete W[i];
    delete VMC;
    delete W;
    delete order;
}

//      Johnson's rule for use by the CDS and RA heuristics routines...

void Johnson(float ** VMC,int * order){

    int *U= new int[NJOBS],*V=new int[NJOBS],u=0,v=0;
    float* PU=new float [NJOBS],*PV=new float[NJOBS];
    int i;

    for(i=1;i<=NJOBS;i++){
        if( VMC[0][i]<=VMC[1][i]){
            PU[u]=VMC[0][i];
            U[u++]=i;
        }
        else
        {
            PV[v]=VMC[1][i];
            V[v++]=i;
        }
    }
    if(u!=0){
        Sort(PU,U,u);
        for(i=0;i<u;i++) order[i]=U[i];
    }
    if(v!=0){
        Sort(PV,V,v);
        for(i=0;i<v;i++) order[i+u]=V[v-i-1];
    }
    delete U;
    delete V;
    delete PU;
    delete PV;
}

```

```

//      File 6   Routine for the Ho_Chang Heuristic ...
//      FHO.CPP

// Ho_Chang Heuristics details ....

#include<fstream.h>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
# include "chrom.h"
extern int POP, NJOBS, NMC ;
extern class Job *job;
float ***d, **dr,* factor;
extern chrom ref;

//      Initialize the Ho_Chang Heuristic...
void I_HO(void){
int i,j,k;
    d=new float ** [NMC-1];
    dr= new float * [NJOBS];
    for(i=0;i<NMC-1;i++)
        d[i]=new float *[NJOBS];
    for(i=0;i<NJOBS;i++)
        dr[i]=new float[NJOBS];
    for(i=0;i<NMC-1;i++)
        for(j=0;j<NJOBS;j++)
            d[i][j]=new float[NJOBS];
    factor=new float [NMC-1];

    for(i=0;i<(NMC-1);i++) {
        factor[i]=.9/(NMC-2)*(NMC-i-2) + .1;
    }
    for(k=0;k<NMC-1;k++) {
        for(i=0;i<NJOBS;i++) {
            for(j=0;j<NJOBS;j++) {
                if(i!=j)
                    d[k][i][j]=job[i+1].ptime[k+1]- job[j+1].ptime[k];
            }
        }
    }
    for(i=0;i<NJOBS;i++) {
        for(j=0;j<NJOBS;j++) {
            dr[i][j]=0;
        }
    }
    if(i!=j) for(k=0;k<NMC-1;k++) {
        if(d[k][i][j]<0)
            dr[i][j]+=d[k][i][j]*factor[k]; else
        dr[i][j]+=d[k][i][j];
    }
    for(i=0;i<NMC-1;i++)
        for(j=0;j<NJOBS;j++)
            delete d[i][j];
    for(i=0;i<NMC-1;i++)
        delete d[i];
    delete d;
    delete factor;
}
//      Apply the Ho_Chang Heuristic to improve the job sequence...

void HEURST_HO( chrom & O){
    int i,u,v,a=0,b=NJOBS-1;
    float x,y;
    void step1(int &,int, chrom &);
    void step2(int &,int, chrom &);
    float abs1(float);
    O.cal_stat();
    ref=0;
    while(b>a+2){ x=-1e10; y=1e10;
        for(i=a+1;i<b;i++)
            if(x<dr[O.string[a]-1][O.string[i]-1]){
                x=dr[O.string[a]-1][O.string[i]-1];

```

```

        u=i;
    }
    for(i=a+1;i<b;i++){
        if(y>dr[0.string[i]-1][0.string[b]-1]){
            y=dr[0.string[i]-1][0.string[b]-1];
            v=i;
        }
        if( (x<0) && (y>0) && (absl(x)<=absl(y))) step1(a,u,0);
        else
        if( (x<0) && (y>0) && (absl(x)>absl(y))) step2(b,v,0);
        else
        if(absl(x)>absl(y)) step1(a,u,0);
        else
        step2(b,v,0);
        O.cal_stat();
        if(O>ref){ O=ref;
        }
        else ref=O;
    }
}

void step1(int & a,int u, chrom & o){
int temp;
    a++;
    temp=o.string[a];
    o.string[a]=o.string[u];
    o.string[u]=temp;
}

void step2(int & b,int v, chrom & o){
int temp;
    b--;
    temp=o.string[b];
    o.string[b]=o.string[v];
    o.string[v]=temp;
}

float absl( float a){
    if(a>=0) return(a);
    else return(-1*a);
}

//      File 7  Screen Management and Display routines...
//      BAGCHI.CPP

#include <fstream.h>
#include <conio.h>
#include <stdlib.h>
#include <graphics.h>
#include <math.h>
#include <dos.h>
#include <string.h>
#include "chrom.h"
int or_x, or_y;
extern char option1,option2,option3;
extern Job *job;
void PALMER(chrom &);

void NEH(chrom &);

void RA(chrom &);

void CDS(chrom &);

void Help1();

void Help2();

void Help3();

void Help4();

void Help5();

void SHOW_T();

void Help6();

void READ();

void kill();

```

```

void killho();
extern ifstream fin;
extern int NJOBS,NMC,SEED,GEN,POP;
extern float *doe;
extern float CROSS_PROB, MUT_PROB;
int P,N,R,C;
extern char * file;
void Menu( int a=0);
#define SHADOW 1
#define NO_SHADOW 0
#define Yes 1
#define No 0
#define define Esc 27

/*********************************************
void box(int xl, int yl,int x2,int y2,int bkc,int textc,int shadow)
{
    int i,j;
    textcolor(2);
    textbackground(bkc);
    if(x2!=80 && y2!=25) { for(j=xl; j<=x2 ; j++)
        for(i=yl; i<=y2-1; i++) { gotoxy(j,i); cprintf(" ");}
        for(j=xl; j<=x2-1; j++) { gotoxy(j,y2); cprintf(" ");}
    textcolor(textc);
    for(j=xl+1; j<=x2-1; j++) // hori line
    { gotoxy(j,yl); putch(char(205));
        gotoxy(j,y2); putch(char(205));}
    for(i=yl+1; i<=y2-1; i++) // ver line
    { gotoxy(xl,i); putch(char(186));
        gotoxy(x2,i); putch(char(186));}
        gotoxy(xl,yl); putch(char(201)); // left top
        gotoxy(xl,y2); putch(char(200)); // left bottom
        gotoxy(x2,y1); putch(char(187)); // right top
    if(y2!=25 && x2!=80) { gotoxy(x2,y2); putch(char(188));}
// right bottom
    if(shadow==SHADOW) { textbackground(0);
        for(j=xl+1; j<=x2+1; j++) { gotoxy(j,y2+1); cprintf(" ");}
        for(i=yl+1; i<=y2+1; i++) { gotoxy(x2+1,i); cprintf(" ");}
        for(i=yl+1; i<=y2+1; i++) { gotoxy(x2+2,i); cprintf(" ");}
    }
    gotoxy(xl,yl);
}

void Menu(int h)
{ option1=option2=option3=0;
  if ( h==1 ) goto A;
  if(h==2) goto C1;

A:           system("cls");
  box(2,2,79,25,14,15,0);
  box(5,3,76,23,1,15,0);
  gotoxy( 32,6);

cprintf(" ABOUT This SW..."); window(14,8,68,22);textcolor(11);
cprintf("This software sequences jobs to minimize makespan in a 'static'
Flowshop. It ");
cprintf("uses Palmer, NEH, CDS and RA heuristics and Darwinian as well as
Lamarckian Genetic");
cputs(" Algorithms to solve this NP-hard combinatorial problem.");
  textcolor(12);
window(17,11,64,25);gotoxy(5,4);
cputs("Copyright Nitin Jain and T P Bagchi 1999");
gotoxy(5,5); cputs("User assumes all risks direct or implied"); textcolor(10);
gotoxy(1,8);cputs(" Dept. Of Industrial And Management Engineering");
gotoxy(1,9);cputs("      INDIAN INSTITUTE OF TECHNOLOGY KANPUR      ");
gotoxy(1,10);cputs("          INDIA 208016                  ");

  textcolor(2); gotoxy(4,2);
// cputs("Developed By");  gotoxy(4,5);cputs("Under The Guidance Of");
  window(1,1,80,25);   textcolor(15);
  option1= getch();
  if ( option1==3) { system("cls");
    exit(1); }

}

```

```

B:         kill();
if(option3=='y'||option3=='Y') killho();

A1:
    if (h==1) fin.close();
xA1:
    system("cls");
    box(2,2,79,25,15,15,0);
    box(15,5,65,20,1,15,0);
gotoxy(5,24); textcolor(WHITE+BLINK);cprintf(" ? Help ");
textcolor(15);
    gotoxy(65,24);cprintf("Ctrl-c Quit ");
    gotoxy(30,24); cprintf(" Press @ for directory");

A2:   gotoxy(18,8);
cputs(" Enter Data File Name : ");
{
    int count=0,x;
    static char string[10];
    char str[10];
    for (x=0;x<9;x++) str[x]=' '; str[9]='\0';
    cout<<string; gotoxy(42,8);
    while( (x=getch())!=13)
    {
        if( x==3) { system ("cls"); exit(1);}
        if( x=='?'){ Help4(); goto A1;}
        if( x=='@' )
        system("cls");
        gotoxy( 25,6); cputs(" DIRECTORY CONTENTS :");gotoxy(3,8);
        system("dir *.ga");
        gotoxy(3,8);delline();delline();delline();
        getch(); system("cls");
        goto A1;
    }
if ((x>47 && x<58) || ( x>64 && x<91) ||( x>96 && x<123)|| x=='.')
string[count++]=x;
    string[count]='\0';
}
    if ( x==8) { if (count!=0) string[--count]='\0';}
    gotoxy(42,8);
    cout<<str;
    gotoxy(42,8);
    cout<<(string);
}
file=string;
fin.open( file);
if (! fin) {
    gotoxy(24,17);
    cputs(" File not found, Try again ");
    sleep(1);
    string[0]='\0';
    gotoxy(24,17);
}
cputs( "                                     ");
    goto A2;
}
}

{
    gotoxy(21,12); fin>>NJOBS;
if (! fin) { cout<<"Data File Corrupted"; exit(1);}
fin>>NMC; if (! fin) { cout<<"Data File Corrupted"; exit(1);}
    gotoxy(21,12); cputs("Current Problem :");
    gotoxy (40,13); cputs( "No of Jobs: "); cout<<NJOBS;
    gotoxy (40,14); cputs( "No of Machine: "); cout<<NMC;
    gotoxy(25,19); cputs("PROCEED [Y/N] (Y is default)");
    { char a; a= getch(); if ( a=='n' || a=='N' ){
        fin.close(); goto A1;
    }
    if (a==3) {system("cls");exit(1);}
}
system("cls");

```

```

//      Read processing times of jobs from the input file ...
    {
        job= new Job[NJOBS+1];
    int i;
    for(i=0;i<NJOBS;i++){
        if(! fin)
        {
            cout<<" File '" <<file<<" is corrupted";
            exit(1);}
        job[i+1].readptime();
    }
}

if( h==0) { chrom a;
    for(int i=0;i<NJOBS;i++) a.string[i]=i+1 ;
    PALMER(a); a.cal_stat();P=a.cmax;
    NEH(a); N=a.cmax;
    CDS(a); C=a.cmax;
    RA(a); a.cal_stat(); R=a.cmax;
}

C1:
box(2,2,79,25,15,15,0);
box(8,5,73,21,1,15,0);

gotoxy(36,6);cprintf("MENU ");

gotoxy(32,9); cprintf("1. Standard GA (SGA )");
gotoxy(32,11); cprintf("2. Hybridized GA (HGA)");
gotoxy(32,13); cputs("3. SGA with Mutiple Seeds");
gotoxy(32,15); cputs("4. HGA with Mutiple Seeds");
gotoxy(35,19); cprintf("Press Option # ");
gotoxy(5,24); cprintf(" ? Help (to learn about GA )";
gotoxy(35,24); cprintf("Ctrl-c Quit ");
gotoxy(50,24); cprintf(" Backspace (previous screen) ");
option1=getch();

while(option1!='1' && option1!='2' && option1!='3' && option1!='4' && option1!=8)
{
    if (option1=='?') { Help1(); goto C1;}
    if (option1==3){ system("cls"); exit(1);}
    option1=getch();
}

system("cls");
if ( option1==8) { delete [] job;fin.close(); goto xA1;}
if( option1=='2'||option1=='4'{

C:
box(2,2,79,25,15,15,0);
box(8,4,73,23,1,15,0);
{
gotoxy(15,7);cprintf("INITIALIZE THE POP. BY: "); gotoxy(48,7);
cputs(" MAKESPAN BY HEURISTIC ");
gotoxy(19,9); cprintf("1. Palmer Heuristic."); gotoxy(56,9); cout<<P;
gotoxy(19,11); cprintf("2. NEH Heuristic. "); gotoxy(56,11); cout<<N;
gotoxy(19,13);cprintf("3. CDS Heuristic. "); gotoxy(56,13); cout<<C;
gotoxy(19,15);cprintf("4. RA Heuristic. "); gotoxy(56,15); cout<<R;
gotoxy(34,19); cprintf("Press Option ");
gotoxy(5,24);cprintf(" ? Help ");
gotoxy(35,24);cprintf("Ctrl-c Quit ");
gotoxy(52,24); cprintf(" Backspace (previous screen) ";
}
option2=getch();
while(option2!='1' && option2!='2'&& option2!='3' && option2!='4'&& option2!=8)
{
    if (option2=='?') { Help2(); system("cls"); goto C;}
    if(option2==3) {system("cls");exit(1);}
    option2=getch();
}
system("cls");
if (option2==8) goto C1;

```

```

D:
box(2,2,79,25,15,15,0);
box(8,5,73,21,1,15,0);

gotoxy(20,9);cprintf(" APPLY HO_CHANG HEURISTIC FOR LOCAL CLIMBING?");
gotoxy(27,13);cprintf("(You must enter Y or N): <Y/N> ");
gotoxy(35,19); cprintf(" Press Option ");
gotoxy(5,24);cprintf(" ? Help ");
gotoxy(35,24);cprintf("Ctrl-c Quit ");
gotoxy(52,24); cprintf(" Backspace (previous screen) ");
option3=getch();
while( option3!='Y' && option3!='y'&& option3!='n'&& option3!='N' &&
option3!=8)
{
    if (option3=='?') { Help3();system("cls"); goto D;}
    if (option3==3) { system("cls");exit(1);}
    option3=getch();
}

system("cls");
if (option3==8) goto C;
}

//      Read the values entered for GA parameters...
READ();
}

//*****
//      Routine reads the values of the GA parameters...
void READ(){

system("cls");
int x3=0,x4=0,x5=0,x6=0;

A:   box(2,2,79,25,15,15,0);
box(15,5,69,23,1,15,0);
    gotoxy(8,24);textcolor(WHITE+BLINK);cprintf(" ? Help ");
textcolor(15);
    gotoxy(32,24);cprintf("Ctrl-c Quit ");
    gotoxy(54,24);cprintf(" m Menu ");
    gotoxy(72,24);cprintf("u UP ");
    gotoxy(34,7); cputs("GA PARAMETERS");

A1:  gotoxy(18,10);
    cputs(" ENTER POPULATION SIZE : ");
    if ( x3==0)
    {
        int count=0,x;
        static char string[10],str[10];
cout<<POP;    gotoxy(43,10);
        for (x=0;x<9;x++) str[x]=' '; str[9]=0;
        while( (x=getch())!=13)
        {
            if( x==3) { system ("cls"); exit(1);}
            if( (x>=48 && x<=57) || x=='?' || x=='m' ||x=='M'|| x==8)
            {
                if( x>=48 && x<=57){     string[count++]=x;
                    string[count]='\0';
                }
                if ( x==8){ if (count!=0) string[--count]='\0'; }
                if ( x=='m'||x=='M') goto X;
            }
            if( x=='?'){ Help5(); goto A;}
            gotoxy(43,10);
            cout<<str;
            gotoxy(43,10);
            cout<<(string);
        }
    }

    {int tmp=count-1,coun=count;POP=0;
     while( --coun>=0)
POP+= pow10(tmp-(coun))* ((int)string[coun]-48) ;
}

```

```

        x3=1;
    }
}
x3=1;
}
else {cout<<POP; x3=1; }

gotoxy(18,13);
cputs(" ENTER CROSSOVER PROBABILITY (0.0 to 1.0): ");
if (x4==0)
{
    int count=0,x;
    static char string[10],str[10];
    cout<<CROSS_PROB; gotoxy(61,13);
    for (x=0;x<9;x++) str[x]=' '; str[9]=0;
    while( (x=getch())!=13)
    {
        if( x=='u'||x=='U') { x3=0; goto A1; }
        if( x==3) { system ("cls"); exit(1); }
        if( (x>=48 && x<=57) || x=='?'|| x=='m' ||x=='M'|| x==8||x==46)
        {
            if( (x>=48 && x<=57)|| x==46){string[count++]=x;
                string[count]='\0';
            }
        if ( x==8){ if (count!=0) string[--count]='\0';}
            if ( x=='m'||x=='M') goto X;

            if( x=='?'){ Help5(); goto A; }
            gotoxy(61,13);
            cout<<str;
            gotoxy(61,13);
            if(string[0]!='0')cout<<"0";
            cout<<(string);
        }
    }

xx:   { if(string[0]=='0'){for(int junk=0;junk<9;junk++)
                    string[junk]=string[junk+1];--count; }
        if(string[0]=='0') goto xx;
    }

    if ( string[0]=='.')
    {   int coun=count;           CROSS_PROB=0;
while( --coun>0)
    CROSS_PROB +=(1/ pow10(coun))* ((int)string[coun]-48) ;

        x4=1;
    }   else

        { int tmp=count-1,coun=count;
          CROSS_PROB=0;
while( --coun>=0) {
    CROSS_PROB+= pow10(tmp-(coun))* ((int)string[coun]-48) ;
        }
        x4=1;
    }
        x4=1;
    }   else { x4=1;cout<<CROSS_PROB;
    }

if( CROSS_PROB>1){
        gotoxy(25,20);
        cputs(" Pc can't exceed 1.0. TRY AGAIN");
        sleep(1);
        gotoxy(25,20);
        cputs(" ");
    };
// Default setting for Crossover probability...

        CROSS_PROB=.7;
        gotoxy(18,8);
        x4=0;
        goto A1;
    }

gotoxy(18,16);

```

```

Ab:      cputs(" ENTER MUTATION PROBABILITY (0.0 to 1.0): ");
if (x5==0)
{
    int count=0,x;
    static char string[10],str[10];

    cout<<MUT_PROB;      gotoxy(60,16);
    for (x=0;x<9;x++) str[x]=' '; str[9]=0;
    while( (x=getch())!=13)

    {
        if( x=='u'||x=='U') { x4=0; goto A1; }
        if( x==3) { system ("cls"); exit(1); }
        if( (x>=48 && x<=57) || x=='?'|| x=='m'|| x=='M'|| x==8|| x==46)
        {
            if(( x>=48 && x<=57)|| x==46){   string[count++]=x;
                string[count]='\0';
            }
            if ( x==8){ if (count!=0) string[--count]='\0';}
            if ( x=='m'||x=='M') goto X;

            if( x=='?'){ Help5(); goto A; }
            gotoxy(60,16);
            cout<<str;
            gotoxy(60,16);
            if(string[0]!='0')cout<<"0";
            cout<<(string);
        }
    xy:       { if(string[0]=='0'){for(int junk=0;junk<9;junk++)
    string[junk]=string[junk+1];--count; }
        if(string[0]=='0') goto xy;
    }

    if ( string[0]=='.')
    {   int coun=count;           MUT_PROB = 0;
        while( --coun>0)
    MUT_PROB+=(1/ pow10(coun))* ((int)string[coun]-48) ;
        x5=1;
    }
    else
    {   int tmp=count-1,coun=count;
        MUT_PROB=0;
    while( --coun>=0)
    MUT_PROB+= pow10(tmp-(coun))* ((int)string[coun]-48) ;
        x5=1;
    }
    else { x5=1;cout<<MUT_PROB; }
    if( MUT_PROB>1)
        gotoxy( 25,20);
        cputs(" Pm can't exceed 1.0. TRY AGAIN");
        sleep(1);  gotoxy(25,20);
        cputs(" " );
        MUT_PROB=.001;
        gotoxy(18,16);
        x5=0;
        goto A1;
    }

    gotoxy(18,19);
    cputs(" ENTER NUMBER OF GA GENERATIONS TO RUN: ");
    if (x6==0)
    {
        int count=0,x;
        static char string[10],str[10];

        cout<<GEN;      gotoxy(58,19);
        for (x=0;x<9;x++) str[x]=' '; str[9]=0;
        while( (x=getch())!=13)
        {
            if( x=='u'||x=='U') { x5=0; goto A1; }
            if( x==3) { system ("cls"); exit(1); }
            if( (x>=48 && x<=57) || x=='?'|| x=='m'|| x=='M' || x==8)

```

```

    {
        if( x>=48 && x<=57){      string[count++]=x;
                                    string[count]='\0';
                                }
        if( x==8){ if (count!=0)  string[--count]='\0'; }
        if( x=='m'||x=='M')  goto X;

        if( x=='?'){ Help5();  goto A;}
        gotoxy(58,19);
        cout<<str;
        gotoxy(58,19);
        cout<<(string);
        }
        { int tmp=count-1,coun=count;GEN=0;
        while( --coun>=0)

GEN+= pow10(tmp-(coun))* ((int)string[coun]-48) ;
        x6=1;
        }
        } x6=1;
        } else { x6=1; cout<<x6; }
        if(option1=='1'|| option1=='2'){

Ac:     system("cls");
box(2,2,79,25,15,15,0);
box(15,5,67,20,1,15,0);
gotoxy(5,24);textcolor(WHITE+BLINK);
cprintf(" ?  Help ");textcolor(15);
gotoxy(35,24);cprintf("Ctrl-c Quit   ");
gotoxy(52,24);cprintf(" m Menu ");
gotoxy(65,24);cprintf("u UP   ");
gotoxy(18,8);
cputs(" ENTER Random Seed Value (1 to 999999): ");
{
        int count=0,x;
        static char string[10],str[10];
        cout<<SEED;      gotoxy(58,8);
        for (x=0;x<9;x++) str[x]=' '; str[9]=0;
        while( (x=getch())!=13)
        {
            if( x=='u'||x=='U') { x6=0; goto A;}
            if( x==3) { system ("cls"); exit(1);}
            if( (x>=48 && x<=57) || x=='?'|| x=='m' ||x=='M'|| x==8)
            {
                if( x>=48 && x<=57){      string[count++]=x;
                                            string[count]='\0';
                                        }
                if( x==8){ if (count!=0)  string[--count]='\0'; }
                if( x=='m'||x=='M')  goto X;
            }

if( x=='?'){ Help6();  goto Ac;}
        gotoxy(58,8);
        cout<<str;
        gotoxy(58,8);
        cout<<(string);
        }
        { int tmp=count-1,coun=count; SEED=0;
        while( --coun>=0)
SEED+= pow10(tmp-(coun))* ((int)string[coun]-48) ;
        }
        }
        goto Y;
    }

X:     { system("cls");Menu(2); goto z;}
Y:
system("cls");
box(2,2,79,25,15,15,0);
box(15,5,65,23,1,15,0);
gotoxy(68,17); cputs("Ps = "); cout<<POP;
gotoxy(68,18); cputs("Pc = "); cout<<CROSS_PROB;
gotoxy(68,19); cputs("Pm = "); cout<<MUT_PROB;
gotoxy(18,10);

```

```

cputs(" Best Makespan Found So Far :");
gotoxy(18,12);
cputs(" Average Makespan           :");
gotoxy(18,14);
cputs(" Worst Makespan            :");
gotoxy(18,8);
cputs("                           Current Gen #:");
gotoxy(5,21);
gotoxy(21,3);
if( option1=='2'){
switch( option2) {
case '1' : cputs(" GA INITIALIZED BY PALMER HEURISTIC "); break;
case '2' : cputs(" GA INITIALIZED BY NEH HEURISTIC "); break;
case '3' : cputs(" GA INITIALIZED BY CDS HEURISTIC "); break;
case '4' : cputs(" GA INITIALIZED BY RA HEURISTIC "); break;
}
}
if( option3=='Y' || option3=='y') {
gotoxy(25,4);cputs(" AND HYBRIDIZED BY HO_CHANG HEURISTIC... ");
}
}
else { gotoxy(35,3);     cputs("STANDARD GA"); }
}
else {
system("cls");
box(2,2,79,25,15,15,0);
box(12,10,65,14,1,18,1); gotoxy(14,12);textcolor(YELLOW+BLINK);
cputs(" JUST A MINUTE PLEASE ... COMPUTER IS THINKING!!! ");
box(22,16,58,18,8,10,0);
gotoxy(24,17);cputs("0%");
}
}
z:
}
// Routine to provide help ...
/*********************************************
void Help1(){
system("cls");
box(2,2,79,25,10,15,0);
box(5,3,76,22,1,15,0);
gotoxy( 35,4);
cprintf("HELP");
cprintf(" Standard GA is the pure Darwinian GA described by Holland. ");
cprintf(" Hybridized GA (HGA) combines Darwinian-Lamarckian concepts of ");
cprintf(" evolution. HGA is expected to perform more effectively and");
cprintf(" efficiently than SGA.");
cprintf(" Press any key ");
getch();
system("cls");
}

/*********************************************
void Help2(){
system("cls");
box(2,2,79,25,10,15,0);
box(5,3,76,22,1,15,0);
gotoxy( 35,4);
cprintf("HELP");
cprintf(" To provide a good start to GA, use solution-generating heuristics.");
cprintf(" The heuristics have been mentioned in the menu, in the order");
cprintf(" of their performance.");
cprintf(" Press any key ");
getch();
system("cls");
}

/*********************************************
void Help3(){
system("cls");
box(2,2,79,25,10,15,0);
}

```

```

        box(5,3,76,22,1,15,0);
        gotoxy( 35,4);
        cprintf("HELP");
gotoxy(8,6);cprintf(" To add Lamarckism to GA, Ho-Chang heuristic is");
gotoxy(8,7);cprintf(" invoked whenever the GA does not improve the solution for
5");
gotoxy(8,8);cprintf(" successive generations.");
        gotoxy(8,9); cprintf("");
        gotoxy(33,19); cprintf(" Press any key ");
        getch();
        system("cls");
    }

/*****************************************/
void Help4(){
system("cls");
box(2,2,79,25,10,15,0);
box(5,3,76,22,1,15,0);
        gotoxy( 35,4);

        cprintf("HELP");
gotoxy(8,6); cprintf(" The data file contains the processing times of jobs on
machines. ");
gotoxy(8,8); cprintf(" Following should be the format of data file for n job
and ");
gotoxy(8,9); cprintf(" m machine problem:");
        gotoxy(13,11); cprintf(" n m");
        gotoxy(13,12); cprintf(" J11 J12 J13 J14 .....J1m");
        gotoxy(13,13); cprintf(" J21 J22 J23 J24 .....J2m");
        gotoxy(13,14); cprintf(" . . . . . . . . ");
        gotoxy(13,15); cprintf(" Jn1 Jn2 Jn3 Jn4 .....Jnm");
gotoxy(10,17); cprintf(" Where, Jij denotes 'processing time' of ith job on jth
machine.");
        gotoxy(33,20); cprintf(" Press any key ");
        getch();
        system("cls");
    }

/*****************************************/
void Help5(){
system("cls");
box(2,2,79,25,10,15,0);
box(5,3,76,25,1,15,0);           gotoxy(34,7); cputs("GA PARAMETERS");
        window( 6,9,75,24);
cputs("The populations size is the no. of chromosomes ga starts its
processing");
cputs(" with. Crossover and mutation probabiliteis are another GA parameters");
cputs(" that contol the GA search."); gotoxy(1,5);
cputs("Optimum values of these operators are problem specific but the");
cputs("default values set here may not be a bad choice. "); gotoxy(1,10);
cputs("No. of generation fixes the termination criterion i.e. no of ");
cputs("generations the GA should process up to. The termination criterion");
cputs(" depends upon the size of problem. For a large problem GA should ");
cputs(" be allowed to run for longer period (generations) and vice-versa.");
cputs(" Though there is no harmin allowing the GA to run longer if");
cputs(" computer time is not a condition.");
        window(1,1,80,25);
        getch(); clrscr();
    }

/*****************************************/
void Help6(){
system("cls");
box(2,2,79,25,10,15,0);
box(5,3,77,23,1,15,0); gotoxy(37,7); cputs("SEED #");
        window( 6,9,76,24);
cputs("A different Seed no. generates a different stream of random numbers
and");
cputs(" thus control the starting point and by some extent direction of GA ");
cputs("search. With different seeds GA search may end up at differnt");
cputs(" solutions.But a good choice of genetic parameters ( population size,");
}

```

```

cputs("probability of crossover and mutation) nullifies the effect of Seed");
cputs(" no. and make the GA robust i.e. GA ends up at equally good solutions
each time, ");
cputs(" irrespective of the Seed no.");
getch(); window(1,1,80,25);
system("cls");
}

// File 8 Routine to display the results graphically on screen...
// GRAPH.CPP

#include <iostream.h>
#include <fstream.h>
#include <conio.h>
#include <stdlib.h>
#include <graphics.h>
#include<dos.h>
extern char option1,option2,option3;
extern int P,R,C,N;
extern char option1,option2;
void graph(void)
{
    float SLE;
if (option1=='1' )SLE=.80; else SLE=.82;
    int a,b,errorcode;
    a=DETECT;
    initgraph(&a,&b,"..\\bgi");
    errorcode = graphresult();
    if(errorcode != 0 )
    {
        cout << " Graphics Error!!!!!!,add *.bgi files in current dir.";
        exit(0);
    }
    ifstream input("data1");
    setbkcolor(15);
    setcolor(8);
    int or_x, or_y;
    char str[10],POP[6];
    char CS[6],MUT[6];
    or_x = 120;
    or_y = 380;
    setlinestyle(0,0,3);
    line(or_x,or_y,620,or_y);
    line(or_x,30,or_x,or_y);
    settextstyle(DEFAULT_FONT, VERT_DIR, 1);
    outtextxy(or_x-100,or_y-250,"Makespan ----> ");
    settextstyle(DEFAULT_FONT, HORIZ_DIR, 1);
    outtextxy(or_x+200,or_y+40,"Generations ----> ");
    setcolor(1); outtextxy(or_x-15,or_y+60,"POPULATION = ");
    input>>POP>>CS>>MUT;
    input>>POP>>CS>>MUT;

    int maxgen, maxspan, minspan, incre,w1,w,av,av1;
    input >> maxgen;
    input >> maxspan; input>>w1; input>>av1;
    input >> minspan;

    outtextxy(or_x+84,or_y+60,POP);
    outtextxy(or_x+145,or_y+60,"PC = ");
    outtextxy(or_x+185,or_y+60,CS);
    outtextxy(or_x+250,or_y+60,"Pm = ");
    outtextxy(or_x+286,or_y+60,MUT);
    outtextxy(or_x+350,or_y+60,"Generations =");
    itoa(maxgen, str, 10); outtextxy(or_x+460,or_y+60,str);
    setcolor(8); setlinestyle(0,0,1);
    incre = int(maxgen/10+0.5);
    for(int i=0; i<=10; i++)
    {
        outtextxy(or_x+i*48,or_y,"|");
        itoa(i*incre, str, 10);
        outtextxy(or_x+i*48,or_y+20,str);
    }
}

```

```

outtextxy(or_x-10,or_y-25,"--");
itoa(int(SLE*maxspan), str, 10);
outtextxy(or_x-40,or_y-25,str);

w = int( (or_y -170) + ((155.0*(maxspan-w1))/(maxspan-SLE*maxspan)) );
outtextxy(or_x-10,w,"--");
itoa(w1, str, 10);
outtextxy(or_x-40,w,str);

av = int( (or_y -170) + ((155.0*(maxspan-av1))/(maxspan-SLE*maxspan)) );
outtextxy(or_x-10,av,"--");
itoa(av1, str, 10);
outtextxy(or_x-40,av,str);

input.close();

input.open("data2");
int x1,x2,y1,y2,y3,y4,gen,span,w0,av0;

x1 = or_x; y1 = or_y-170;

while (input)
{
    input >> gen;
    input >> span;   input>>w0; input>>av0;
    x2 = int( or_x+4 + (500.0*gen)/maxgen );
y2 = int( (or_y -170) + ((155.0*(maxspan-span))/(maxspan-SLE*maxspan)) );
y3 = int( (or_y -170) + ((155.0*(maxspan-w0))/(maxspan-SLE*maxspan)) );
y4 = int( (or_y -170) + ((155.0*(maxspan-av0))/(maxspan-SLE*maxspan)) );

    setcolor(1);           line(x1,y1,x2,y2);   circle(x1,y1,1);
    setcolor(5);           line(x1,w,x2,y3);   circle(x1,w,1);
    setcolor(2);           line(x1,av,x2,y4);   circle(x1,av,1);
    setcolor(8);

    delay(5);
    x1 = x2; y1 = y2; w=y3;av=y4;
}
setcolor(1); outtextxy(or_x+320,40,"GA BEST MAKESPAN :");
itoa(span,str,10);outtextxy(or_x+470,40,str);
setcolor(4); outtextxy(or_x+320,55,"GA WORST MAKESPAN:");
itoa(w0,str,10);outtextxy(or_x+470,55,str);
setcolor(2); outtextxy(or_x+320,70,"GA AVG. MAKESPAN :");
itoa(av0,str,10);outtextxy(or_x+470,70,str);
setcolor(1); outtextxy(or_x-10,y1,"--");
itoa(minspan, str, 10);
outtextxy(or_x-40,y1,str); outtextxy(or_x-75,y1,"BEST");
y1 = int( (or_y -170) + ((175.0*(maxspan-P))/(maxspan-SLE*maxspan)) );
setcolor(3); outtextxy(or_x-10,y1,"--");
itoa(P, str, 10);
outtextxy(or_x-40,y1,str);outtextxy(or_x-75,y1,"PAL");
y1 = int( (or_y -170) + ((155.0*(maxspan-N))/(maxspan-SLE*maxspan)) );
setcolor(3); outtextxy(or_x-10,y1,"--");
itoa(N, str, 10);
outtextxy(or_x-40,y1,str); outtextxy(or_x-75,y1,"NEH");
y1 = int( (or_y -170) + ((155.0*(maxspan-C))/(maxspan-SLE*maxspan)) );
setcolor(3); outtextxy(or_x-10,y1,"--");
itoa(C, str, 10);
outtextxy(or_x-40,y1,str); outtextxy(or_x-75,y1,"CDS");
y1 = int( (or_y -170) + ((155.0*(maxspan-R))/(maxspan-SLE*maxspan)) );
setcolor(3); outtextxy(or_x-10,y1,"--");
itoa(R, str, 10);
outtextxy(or_x-40,y1,str); outtextxy(or_x-75,y1,"RA");
input.close(); input.open("x"); if(input){
    int a,b,c;
    input>>a>>b>>c;
    setcolor(12); outtextxy(or_x-10,y1,"--");
    itoa(a, str, 10);
y1 = int( (or_y -170) + ((155.0*(maxspan-a))/(maxspan-SLE*maxspan)) );
setcolor(12); outtextxy(or_x-10,y1,"--");
    itoa(a, str, 10);
}

```

```

        outtextxy(or_x-40,y1,str);outtextxy(or_x-75,y1,"B.R.");
y1 = int( (or_y -170) + ((155.0*(maxspan-b))/(maxspan-SLE*maxspan)) );
    setcolor(12);          outtextxy(or_x-10,y1,"--");
    itoa(b, str, 10);
    outtextxy(or_x-40,y1,str); outtextxy(or_x-75,y1,"W.R.");
y1 = int( (or_y -170) + ((155.0*(maxspan-c))/(maxspan-SLE*maxspan)) );
    setcolor(12);          outtextxy(or_x-10,y1,"--");
    itoa(c, str, 10);
    outtextxy(or_x-40,y1,str); outtextxy(or_x-75,y1,"Av.R.");
}
outtextxy(125,or_y+80," B.R ='Random Best'    W.R.= 'Random Worst'   Av.R.= Random
Avg ");
outtextxy(125,or_y+90,"           ( In trials equal to populaton size)");
    setcolor(8);
    if( option1=='2'){
        switch( option2) {
case '1' : outtextxy(or_x+115,or_y-370," INITIALIZED BY PALMER HEURISTIC ");
break;
case '2' : outtextxy(or_x+115,or_y-370," INITIALIZED BY NEH HEURISTIC "); break;
case '3' : outtextxy(or_x+115,or_y-370," INITIALIZED BY CDS HEURISTIC "); break;
case '4' : outtextxy(or_x+115,or_y-370," INITIALIZED BY RA HEURISTIC "); break;
        }
if( option3=='Y' || option3=='y') {
outtextxy(or_x+150,or_y-350," AND HO_CHANG ASSISTED");
}
    }
else {     outtextxy(or_x+188,or_y-370,"DARWINIAN GA");}
getch();
system("cls");
closegraph();
}

//      chrom.h  ->NOT TO BE INCLUDED IN THE C++ PROJECT, BUT SHOULD EXIST
//      File 9      in the SOURCE DIRECTORY CONTAINING THE *.CPP Program files.
//      DECLARATION OF CLASSES :

#ifndef _RANDOM
#define _RANDOM
class chrom;
class random {
private:
    float r_var(void);           //generate a random variable in between 0-1
public:
int zero_one(float prob=.5);
//      generate 1 with prob. p and 0 with (1-p)
int crossover(int length,float prob=.8);
//      generate the position of crossover in a pair of chromosomes
int r_wheel(chrom * );
int r_wheel(int a);
    random();
    void change_seed( int seed=0);
//      Initialize the random varible generator
    };
#endif

#ifndef _CHROM
#define _CHROM
class random;
class chrom{
public:
float fitness_value();           // calculate the fitness value

int * string;
static int njobs;
static int nmc; // no. of variables
static class random ra;
static int pop;           // population

```

```

static int count;
float cmax;
float s_prob;           // prob. of being reproduced
float cum_prob;         // cumulative prob. of being reproduced

void schedule(int njobs);
friend void cal_sprob(chrom *, int=0);
friend void cal_cumprob(chrom *, int=0);
float max(float a, float b);
chrom();
void cal_stat(int x=njobs);      // calculates statistics
float idle_time(int x=njobs);
void show(int x=njobs); // shows statistics
~chrom();
void operator=(chrom &);
int operator>(chrom &);
};

#
#endif _JOB
#define _JOB
class Job{

public:
    int nmc;
    float *ptime;
    float *starting;
    float *finishing;
    void readptime();
    void readptime(char *);

    int idle_time;
    void cal_stat();
    Job();
    friend void show_stat(Job *);
    ~Job(); };

#
#endif

```

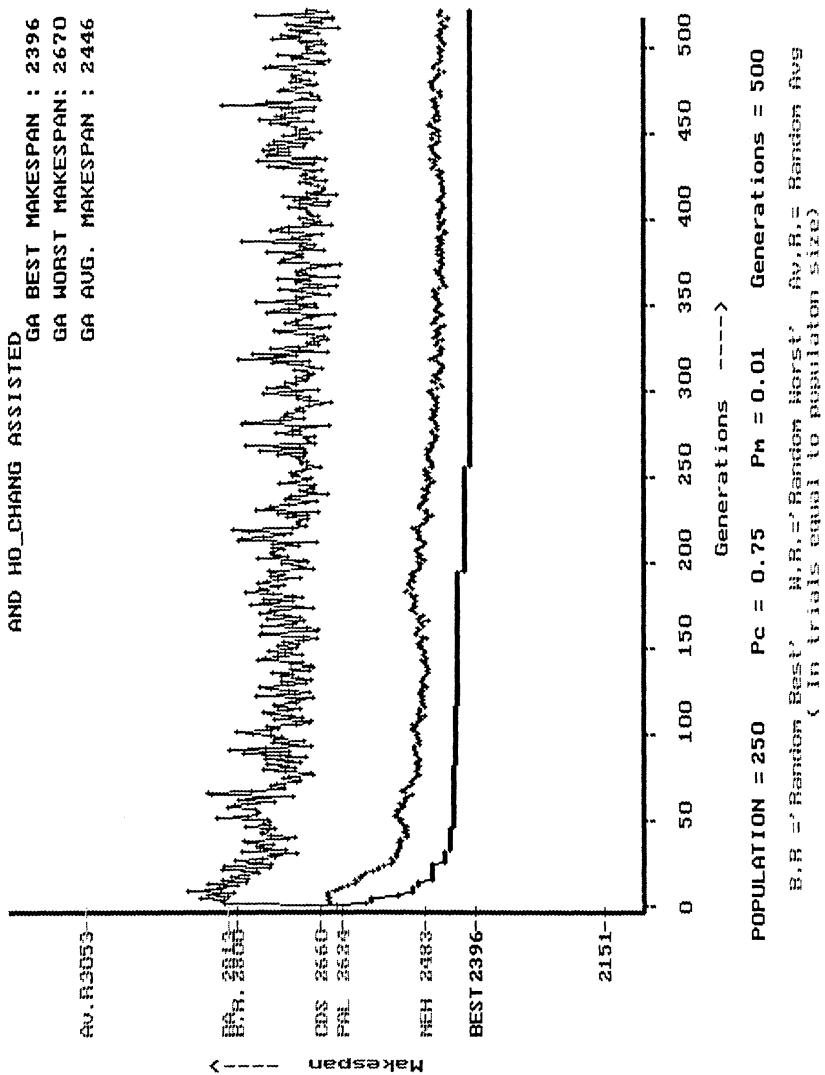
SAMPLE FLOWSHOP DATA FILE

30 15
74 72 54 57 52 60 4 8 40 8 85 45 74 67 48
99 77 58 50 31 67 19 96 93 29 27 6 85 22 48
15 10 85 2 92 53 60 63 11 94 44 71 19 99 94
63 18 44 30 80 94 63 28 50 55 78 83 8 68 65
94 42 20 92 73 62 45 86 76 11 80 53 29 3 70
16 15 97 30 7 31 82 10 28 13 63 55 24 26 49
51 57 19 87 81 17 8 27 93 72 1 19 30 80 86
18 46 17 12 54 54 90 52 69 82 47 96 90 14 12
77 29 52 40 83 53 44 49 87 60 2 88 26 18 30
19 95 57 94 93 19 36 14 82 74 94 7 90 40 39
25 28 72 89 5 87 15 87 14 20 24 91 93 41 36
92 98 56 35 64 15 95 22 67 61 12 98 73 6 10
47 80 88 77 77 60 63 66 8 10 63 74 90 1 56
38 84 99 21 13 73 0 26 68 99 9 72 42 43 27
77 65 38 88 95 9 13 13 42 55 51 36 27 78 12
22 81 30 45 63 94 44 78 98 57 26 85 61 82 71
6 3 43 96 51 39 79 28 50 27 49 30 73 53 85
32 28 61 62 18 76 92 50 5 26 1 53 33 79 17
28 97 87 10 79 35 55 72 98 32 2 58 87 86 12
16 73 45 90 76 86 75 38 58 49 61 90 23 92 24
47 8 30 52 53 96 26 76 18 65 32 18 35 71 36
65 90 73 28 34 58 20 10 46 94 38 34 0 17 72
6 62 42 42 95 26 90 83 82 54 47 44 80 22 2
39 12 28 15 77 22 46 17 19 18 83 5 28 55 96
4 75 57 1 24 46 99 43 17 22 36 22 12 22 38
14 97 19 81 92 44 66 92 71 66 43 61 11 42 82
38 55 10 76 22 61 40 59 46 39 33 2 92 81 99
82 38 7 12 2 28 88 89 2 93 88 60 60 53 82
68 55 77 51 34 25 35 45 38 77 96 3 27 23 43
86 4 95 81 63 51 73 80 1 10 11 47 87 80 54

INITIALIZED BY PALMER HEURISTIC

AND HO-CHANG ASSISTED

GA BEST MAKESPAN : 2396
GA WORST MAKESPAN: 2670
GA AUG. MAKESPAN : 2446



SAMPLE GRAPHICAL OUTPUT SHOWING THE CONVERGENCE
OF THE HYBRID GA ALONG WITH MAKESPANS GIVEN BY
VARIOUS HEURISTICS

Glossary

Abstraction A method to solve hard problems. In it we separate out the important features of the problem from the unimportant ones that drown the process (the mechanism of what causes what and how) underlying the problem.

Acquired Characteristics Traits, mannerisms or constitutional modifications that organisms pick up during their lifetime. An example is strong right biceps that an ironsmith develops. These re-shape their **phenotype**. Acquiring such characteristics may modify the organism's proteins but not its genetic material.

Adaptation (in evolution) Any change in the structure or functioning of an organism that makes it better suited to its environment. Natural selection of *inheritable adaptations* ultimately leads to the development of new species. Adaptation to a particular environment tends to diminish an organism's ability to adapt to any sudden change in that environment.

Adaptive Plan Any scheme by which an individual or a succession of generations makes itself better suited to its environment. Such schemes may involve natural selection, genetic processes such as mutation, crossover, inversion, learning, sharing, niche formation, speciation, etc.

Allele One of two or more alternative forms of a single **gene** locus. Different alleles of a gene each have a unique **nucleotide** sequence, and their activities are all concerned with the same biochemical and developmental process, although their individual phenotypes may differ. There are usually two alleles of any one gene (one from each parent), which occupy the same relative position (locus) on homologous chromosomes. One allele is often dominant to the other (known as the recessive), i.e. it determines which aspects of the particular characteristic the organism will display. Within a population there may be several alleles of a gene; each has a unique nucleotide sequence.

Artificial Intelligence A scheme artificially created to tackle hard or large search or decision making problems incorporating the faculty of understanding and comprehension including learning and reasoning, with the objective of solving problems. Often used when direct, deductive methods become ineffective.

Base Pairing The chemical linking of two complementary nitrogenous bases in DNA and in certain types of RNA molecules. Of the four such bases in DNA, adenine pairs with thymine and cytosine with guanine. In RNA, thymine is replaced by uracil. Base pairing is responsible for holding together the two strands of a DNA molecule to form a *double helix* and for faithful reproduction and reading of the genetic code. The links between bases take the form of hydrogen bonds.

Central Dogma The basic belief originally held by molecular geneticists that flow of genetic information can only occur from DNA to RNA to proteins. It is now known, however, that information contained within RNA molecules of viruses can also flow back to DNA.

Chromatid A threadlike strand formed from a chromosome during the early stages of cell division. Each chromosome divides along its length into two chromatids, which are at first held together. They separate completely at a later stage. The DNA of the chromosome reproduces itself exactly so that each chromatid has the complete amount of DNA and becomes a daughter chromosome with exactly the same genes as the original chromosome from which it was formed.

Chromatin The substance of which eukaryotic chromosomes are composed. It consists of proteins, DNA, and small amounts of RNA. In a metabolically inactive nucleus, chromatin is mainly in a condensed form, heterochromatin, but in an active nucleus most of the chromatin is in an expanded form, euchromatin. *m*-RNA molecules are formed in euchromatic regions.

Chromosome The genetic material of the cell, complexed with protein and organized into a number of linear structures. The term literally means "colored body" and is a threadlike structure found in the nucleus of plant and animal (eukaryotic) cells. Chromosomes are composed of chromatin and carry the genes in a linear sequence; these determine the individual characteristics of an organism. When the nucleus is not dividing, individual chromosomes cannot be identified with a microscope. During nuclear division, the chromosomes contract and, when stained, can be clearly seen under a microscope. Each consists of two chromatids (see also **meiosis**; **mitosis**). The number of chromosomes in each cell is constant for and characteristic of a species. In the normal body cells of diploid organisms the chromosomes occur in pairs (**homologous chromosomes**); in the gamete-forming germ cells, however, the diploid number is halved and each cell contains only one member of each chromosome pair. Thus in humans each body cell contains 46 chromosomes (22 matched pairs and one pair of sex chromosomes) and each germ cell 23. Abnormalities in the number or structure of chromosomes may give rise to abnormalities in the individual; Down's syndrome is one such abnormality.

Chromosomal Theory of Inheritance The theory that chromosomes are the carriers of genes. The first clear formulation of the theory was made by both W S Sutton (1905) and T Boveri (1902), who independently recognized that the transmission of chromosomes from one generation to the next closely paralleled the pattern of transmission of genes from one generation to the next.

Coding Sequence The part of an *m*-RNA molecule that specifies the amino acid sequence of a polypeptide during translation.

Codon A group of three adjacent nucleotides in an *m*-RNA molecule that specifies either one amino acid in a polypeptide chain or the termination of polypeptide synthesis.

Combinatorial Optimization A method in which an optimal solution is found among a finite or a countably infinite number of alternatives. In many such situations the combination of several goals, constraints and resources may result in exponentially growing number of possible solutions. In such cases it may become

difficult or even impossible to find exact solutions in reasonable time. The difficulty of solving a combinatorial problem is expressed as its complexity. See **NP hardness**.

Crossing over An exchange of portions of chromatids between homologous chromosomes. As the chromosomes begin to move apart at the end of the first prophase of meiosis, they remain in contact at a number of points. At these points the chromatids break and rejoin in such a way that sections are exchanged. Crossing over thus alters the pattern of genes in the chromosomes. See **recombination**.

Darwinian Adaptation Any change in the structure or functioning of an organism that occurs through genetic variation and natural selection to make the organism better suited to its environment. Natural selection of *inheritable adaptations* ultimately leads to the development of new species.

Darwinian Fitness The relative reproductive ability of a genotype.

Development The process of regulated growth that results from the interaction of the genome with cytoplasm and the environment. It involves a programmed sequence of phenotypic events that are typically irreversible.

Diploid A eukaryotic cell with two sets of chromosomes.

Elitism A GA selection method first introduced by Kenneth De Jong (1975). It forces the GA to retain some of the best individuals at each generation into the next generation. Such individuals can otherwise be lost if crossover or mutation destroys them. Experience shows that elitism significantly improves the GA's performance.

ENGA (Elitist Nondominated Sorting GA) An enhanced version of NSGA that incorporates elitism. NSGA is somewhat deficient in both on-line and off-line performance because it does not preserve good (Pareto optimal or near-Pareto optimal) solutions found in one generation to the next generation. By contrast, ENGA *consciously preserves* a controlled fraction of the best solutions present in any generation. ENGA has been shown to be statistically superior in discovering Pareto optimal solutions to multiobjective numerical and combinatorial problems.

Eukaryote Organisms that have cells in which genetic material is located in a membrane-bound (or "true") nucleus. Eukaryotes can be unicellular or multicellular.

Evolution A series of slow changes that occur as populations of organisms adapt to their changing surroundings. When resources are limited or the environment changes, organisms undergo a struggle for existence. Subsequently, **natural selection** (a "survival-of-the-fittest" struggle) occurs and organisms that survive are allowed to procreate and pass on their special survival capabilities to their offspring. Thus, according to Darwin, a "better fitted" generation evolves.

Fitness Function In order to mimic the survival-of-the-fittest principle of natural evolution, genetic algorithms use a measure of merit (or fitness) for each solution. Subsequently, GA uses this fitness as the basis to decide whether or not that solution will "survive." This measure of fitness is derived from the problem's objective function ($f(x)$) using a mathematical transformation known as the fitness function. For maximization problems the fitness function normally equals $f(x)$. For minimization problems it equals $1/(1 + f(x))$

Fitness Landscape A representation of all possible genotypes along with their fitness. Crossover and mutation operations are ways of moving a GA population around on the fitness landscape. Fitness landscape reflects the variability of the objective function being optimized. Many classical optimization methods require gradient information about the function being optimized whereas meta-heuristic search-based methods such as GA, simulated annealing or tabu search do not.

Fitness Sharing A necessary condition to induce species formation. Fitness sharing reduces intraspecies (i.e., within the same species) competition. In GA, fitness sharing improves convergence and the quality of the final solution.

Flowshop A processing shop or facility characterized by unidirectional flow of work with a variety of jobs, all being processed sequentially in the same order, in a one-pass manner. Typically, each job is different but every job follows the same one-pass routing through the processing stages (called "machines"). Storage between stages is assumed to be unlimited.

Gametes Mature reproductive cells (sperm or ova) that are specialized for sexual function. Each gamete is haploid (has a single copy of each nuclear chromosome) and fuses with a cell of similar origin but of opposite sex to produce a diploid zygote.

Gene (Mendelian factor) The determinant of a characteristic of an organism. Genetic information is coded in the DNA, which specifies a polypeptide or RNA and is subject to mutational alteration.

Gene Pool The total genetic information encoded in the total genes in a breeding population existing at a given time.

Genetic Algorithms An *evolution strategy* is a random search that uses selection and variation. A genetic algorithm is a parallel random search procedure using sexual reproduction with centralized control. Sexual reproduction is characterized by recombining two parent strings into an offspring. This recombination is called crossover. GA also uses mutation as a search strategy. **Mutation** is based on chance and the progress in search for a single mutation step is almost unpredictable. Mutation introduces variation in the population and induces local search while crossover shuffles substrings contained in a population and promotes global search.

Genetic Code The base-pair information that specifies the amino acid sequence of a polypeptide. The means by which genetic information in DNA controls the manufacture of specific proteins by the cell. The code takes the form of a series of triplets of bases in DNA, from which is transcribed a complementary sequence of codons in messenger RNA (see **transcription**). The sequence of these codons determines the sequence of amino acids during **protein synthesis**. There are 64 possible codes from the combinations of the four bases present in DNA and **m-RNA** and 20 amino acids present in body proteins: some of the amino acids are coded by more than one codon, and some codons have other functions.

Genetic Drift Any change in gene frequency due to chance in a population.

Genetic Engineering (Recombinant DNA Technology) Techniques involved in altering the characters of an organism by inserting genes from another organism into its DNA. This altered DNA (known as recombinant DNA) is usually produced by

gene cloning. Genetic engineering is the alteration of the genetic constitution of cells or individuals by directed and selective modification, insertion, or deletion of an individual gene or genes. In some cases, novel gene combinations are made by joining DNA fragments from different organisms. Thus, genetic engineering has many applications, ranging from the commercial production of hormones, vaccines, etc., to the creation of transgenic animals and crop plants in agriculture.

Genetics The science of heredity that involves the structure and function of genes and the way genes are passed from one generation to the next.

Genome The total amount of genetic material in a chromosome set; in eukaryotes, this is the amount of genetic material in the haploid set of chromosomes of the organism.

Genotype The complete genetic makeup of an organism.

Haploid A cell or an individual with one copy of each nuclear chromosome.

Hereditary Trait A characteristic under control of the genes that is transmitted from one generation to another.

Heredity The transmission of characteristics from parents to offspring via the chromosomes; first studied by Mendel who derived a series of laws governing it.

Heritability The proportion of phenotypic variation in a population attributable to genetic factors.

Inheritance The transmission of particular characteristics from generation to generation by means of the genetic code, which is transferred to offspring in the gametes.

Inheritance of Acquired Characteristics (see Lamarckism)

Inversion A chromosome mutation caused by reversal of part of a chromosome, so that the genes within that part are in inverse order. Inversion mutations usually occur during crossing over in meiosis. In inversion a segment of chromosome is excised and then reintegrated in an orientation 180° from the original orientation. A *point mutation* caused by the reversal of two or more bases in the DNA sequence within a gene.

Job Shop Involves processing of jobs on several machines without any "series" structure (unlike the *flowshop*). The facility produces goods according to pre-specified process plans under domain-dependent (technological) and commonsense constraints. Can produce several different parts when each part may have one or more process plans.

Lamarckism One of the earliest superficially plausible theories of evolution, proposed by the French biologist Jean-Baptiste de Lamarck (1744-1829) in 1809. He suggested that changes in an individual are acquired during its lifetime, chiefly by increased use or disuse of organs in response to "a need that continues to make itself felt," and that these changes are inherited by its offspring. Thus the long neck and limbs of a giraffe are explained as having evolved by the animal stretching its neck to browse on the foliage of trees. This so-called *inheritance of acquired characteristics*

has never unquestionably been demonstrated to occur and the theory was largely displaced by Darwinism. Lamarckism is also incompatible with the Central Dogma of molecular biology.

Local Search Heuristics It is generally believed that NP-hard problems cannot be solved within polynomially bounded computation times. Approximation algorithms including local search heuristics can find near-optimal solutions here. A local search algorithm starts off with an initial solution and then continually tries to find better solutions by searching neighborhoods. One version of local search is iterative improvement, which searches its neighborhood for a solution of lower cost. If such a solution is found, it replaces the current solution. If not, it returns the current solution, which is then called a *locally optimal* solution. Linear programming and genetic algorithms are local search methods.

Meiosis Two successive nuclear divisions of a diploid nucleus that result in the formation of haploid gametes having *one-half* the genetic material of the original cell. Such cell division gives rise to four reproductive cells (**gametes**) each with half the chromosome number of the parent cell. Two consecutive divisions occur. In the first, homologous chromosomes become paired and may exchange genetic material (see **crossing over**) before moving away from each other into separate daughter nuclei. This is the actual reduction division because each of the two nuclei so formed contains only half of the original chromosomes. The daughter nuclei then divide by mitosis and four haploid cells are produced.

Mendelian Population An interbreeding group of individuals sharing a common gene pool; the basic unit of study in population genetics.

Mitosis The division of a cell to form two daughter cells each having a nucleus containing the *same number and kind* of chromosomes as the mother cell. The changes during divisions are clearly visible with a light microscope. Each chromosome divides lengthwise into two chromatids, which separate and form the chromosomes of the resulting daughter nuclei. The process has four stages, prophase, metaphase, anaphase, and telophase, which merge into each other. Mitotic divisions ensure that all the cells of an individual are genetically identical to each other and to the original fertilized egg.

MOGA (Multi-Objective Genetic Algorithm) Developed by Fonseca and Fleming (1993) for finding Pareto optimal solutions to multiobjective problems. MOGA uses a solution ranking procedure to suitably position the different solutions relative to each other to impact their chance of selection. The population is sorted for nondominance and all nondominated solutions are assigned rank 1. The rest of the population is ranked by checking nondominance. MOGA sometimes creates large selection pressure and leads to premature convergence.

m-RNA Special RNA molecules that specify the amino acid sequences of proteins which are important structural and functional components of cells.

Multi-criteria Scheduling Allocating limited resources over time among parallel and sequential activities in a factory, hospital, university, etc. so as to optimize multiple management objectives simultaneously. Such multiple objectives may include processing jobs the fastest way possible, maximize the utilization of

expensive capital equipment, labor and utilities, minimize changes and setup, provide uninterrupted supply of goods to customers, etc.

Mutagen An agent that causes an increase in the number of mutants (see mutation) in a population. Mutagens operate either by causing changes in the DNA of the genes, so interfering with the coding system, or by causing chromosome damage. Various chemicals (e.g. colchicine) and forms of radiation (e.g. X-rays) have been identified as mutagens.

Mutant A gene or an organism that has undergone a heritable change, especially one with visible effects (i.e. the change in genotype is associated with a change in phenotype). See mutation.

Mutation Any detectable and heritable change in the genetic material not caused by genetic recombination. Mutation is a sudden random change in the genetic material of a cell that may cause it and all cells derived from it to differ in appearance or behavior from the normal type. An organism affected by a mutation (especially one with visible effects) is described as a mutant. Somatic mutations affect the nonreproductive cells and are therefore restricted to the tissues of a single organism but germ-line mutations, which occur in the reproductive cells or their precursors, may be transmitted to the organism's descendants and cause abnormal development. Mutation occurs naturally at a low rate but this may be increased by radiation and by some chemicals (see mutagen). Most are point (or gene) mutations, which consist of invisible changes in the DNA of the chromosomes, but some (the chromosome mutations) affect the appearance or the number of the chromosomes. An example of a chromosome mutation is that giving rise to Down's syndrome.

In genetic algorithms, mutation is a key operation to facilitate the conduction of local search around a current solution. Mutation creates a new solution in the neighborhood of a current solution by introducing a small change in some aspect of the current solution. If binary GA coding is used, a single bit in a string may be altered from 0 to 1 or 1 to 0 with a small probability, creating a new solution.

Natural Selection The process that, according to Darwinism, brings about the evolution of new species of animals and plants. Darwin noted that the size of any population tends to remain constant despite the fact that more off-spring are produced than are needed to maintain it. He also saw that variations existed between individuals of the population and concluded that disease, competition, and other forces acting on the population eliminated those individuals less well adapted to their environment. The survivors would pass on any inheritable advantageous characteristics (i.e. characteristics with survival value) to their off spring and in time the composition of the population would change. Over a long period of time this process could give rise to organisms so different from the original population that new species are formed.

Neo-Darwinism (modern synthesis) The current theory of the process of evolution, formulated between about 1920 and 1950, that combines evidence from classical genetics with the Darwinian theory of evolution by natural selection (see Darwinism). It makes use of modern knowledge of genes and chromosomes to explain the source of the genetic variation upon which selection works. This aspect was unexplained by traditional Darwinism.

Neo-Lamarckism Any of the comparatively modern theories of evolution based on Lamarck's theory of the inheritance of acquired characteristics (see [Lamarckism](#)). These include the unfounded dogma of lysenkoism and the recent controversial experiments on the inheritance of acquired immunological tolerance in mice.

Niche It describes the role that a particular population (i.e., members of a single species) plays within the ecosystem. It is the particular way by eating habits, predatorialness, environment etc. by which the niche-forming organism fits into the ecosystem. In genetic algorithms niche formation is used to promote fitness sharing among neighboring solutions.

NP Hardness An optimization problem is said to be NP-hard if it is impossible to find an optimal solution to the problem without the use of an essentially enumerative algorithm. Computation times here increase exponentially with problem size.

NSGA (Nondominated Sorting Genetic Algorithm) A multiobjective problem solving GA created by Srinivas and Deb (1995) that uses niche formation and nondominated sorting of solutions in every generation to ensure that solutions on or near the Pareto optimal front get preference in selection for reproduction into the next generation. Parameters p_s , p_m , p_c and σ_{share} are used (see [parameterization](#)). The final results produced are a set of solutions to the multiobjective problem that are Pareto optimal.

Nucleotide An organic compound consisting of a nitrogen-containing purine or pyrimidine base linked to a sugar (ribose or deoxyribose) and a phosphate group. DNA and RNA are made up of long chains of nucleotides (i.e. polynucleotides). Compare nucleoside.

Nucleus (of a Cell) The large body embedded in the cytoplasm of all plant and animal cells (but not bacterial cells) that contains the genetic material, DNA. The nucleus functions as the control center of the cell. It contains a viscous sap (nucleoplasm) and is bounded by a double membrane (the nuclear membrane or envelope), which is perforated by many nuclear pores for the exchange of material between the nucleus and cytoplasm. When the cell is not dividing, the genetic material is dispersed in the sap as chromatin; in dividing cells the chromatin is organized into chromosomes.

Off-line Convergence The capability of an algorithm to consistently reach the true or global optima rather than getting trapped in some local optima along the way.

On-line Convergence A measure that indicates how rapidly an optimization algorithm improves the average quality of the solutions.

Open shop A processing shop similar to the job shop except that there is no *a priori* order of the operations to be done within a single job. Customized (made-to-order) automotive assembly represents an open shop operation.

Parameterization (GA) The selection (specification) or adaptive adjustment of the probability of mutation (p_m), probability of crossover (p_c), population size (p_s), fitness sharing neighborhood radius (σ_{share}), elite fraction (ε), etc. to ensure good off-line as well as on-line convergence of the GA, regardless of the composition of the starting population.

Pareto optimality Multiobjective optimization problems with conflicting objectives typically do not possess a single unique optimal solution. For such problems a set of solutions often exists in which no increase can be obtained in any of the objectives without causing a simultaneous decrease in at least one of the remaining objectives. Such solutions are saddle points in the decision space and are called *Pareto optimal*.

Phenotype The observable properties of an organism that are produced by the genotype, the dominance relationships between the alleles, and by the interaction of the genes with the environment.

Population A group of interbreeding individuals that share set of genes. Members of a single species.

Population Genetics A branch of genetics that describes in mathematical terms the consequences of Mendelian inheritance on the population level.

Protein Proteins are a group of high-molecular weight, nitrogen-containing organic compounds (amino acids) of complex shape and composition. The structure, function, development, and reproduction of an organism depends on the properties of the proteins present in each cell and tissue of the organism. When a protein is needed in a cell, the *genetic code* for that protein's amino acid sequence must be read from the DNA present in the cell and processed into the finished protein. If this code is absent, the related protein cannot be synthesized.

Protein Synthesis The process by which living cells manufacture proteins from their constituent amino acids, in accordance with the genetic information carried in the DNA of the chromosomes. This information is encoded in messenger RNA (*m*-RNA), which is transcribed from DNA in the nucleus of the cell (see *genetic code; transcription*); the sequence of amino acids in a particular protein is determined by the sequence of nucleotides in *m*-RNA. At ribosomes the information carried by *m*-RNA is translated into the sequence of amino acids of the protein (see *translation*).

Recombinant Chromosome A chromosome that emerges from meiosis with a combination of genes different from a parental combination of genes.

Recombinant DNA DNA that contains genes from different sources that have been combined by the techniques of genetic engineering rather than by breeding experiments. Genetic engineering is therefore also known as recombinant DNA technology.

Recombinant DNA Molecule A new type of DNA sequence that has been constructed or engineered in the test tube from two or more distinct DNA sequences.

Recombinant DNA Technology A collection of experimental procedures that allow molecular biologists to splice DNA fragment from one organism into DNA from another organism and to clone the new recombinant DNA molecule. It includes the development and application of particular molecular techniques, such as biotechnology or genetic engineering. This technology is important, for example, in the production of antibiotics, hormones, and other medical agents used in the diagnosis and treatment of certain genetic diseases.

Recombinants Individuals or cells that have nonparental combinations of genes as a result of the processes of genetic recombination.

Recombination Rearrangement of genes that occurs when reproductive cells (gametes) are formed (see *crossing over*). Recombination results in offspring that have a combination of characteristics different from that of their parents. Recombination can also be induced artificially by genetic engineering techniques.

RNA (ribonucleic acid) A usually single-stranded polymeric molecule consisting of ribonucleotide building blocks. RNA is chemically very similar to DNA. Three major types of RNAs exist including the *messenger (m)* RNA, each of which performs an essential role in protein synthesis (*translation*). In some viruses, RNA is the genetic material.

Robust Design A powerful and novel engineering design methodology that creates products and production processes that retain their performance under diverse environmental conditions. The method is experimental and it seeks to reduce the sensitivity of performance to uncontrolled environmental factors through a careful selection of the values of the design parameters. One way to produce a robust design is to apply the "Taguchi method" (Phadke, 1989).

Satisficing Sometimes, the diversity of factors involved in a decision problem may make the finding of the optimal solution prohibitive in cost and/or time. In such cases instead of optimal decisions, *satisficing* decisions are developed by approximation and accepted.

SGA (Simple Genetic Algorithm) A genetic algorithm emulating natural evolution. This was created by Holland (1975) and it incorporated mutation, crossover and natural selection. Subsequently, Lamarckism, fitness sharing, and many other operations have been incorporated.

Sharing Function Sharing is a mechanism to permit similar solutions to coexist by avoiding head-to-head competition, by "sharing" their fitness (and thus reduce self-fitness) with solutions in the neighborhood. The sharing function determines what is a neighborhood and also the degree of sharing that each solution will endure.

Transcription The transfer of information from a double-stranded DNA molecule to a single-stranded RNA molecule. It is also called RNA synthesis.

Translation (protein synthesis) The conversion in the cell of the mRNA base sequence information into an amino acid sequence of a polypeptide.

VEGA (Vector Evaluated Genetic Algorithm) A multiobjective GA created by Schaffer (1984) that does selection *independently for each objective criterion*. Equal-sized subpopulations are created for selection along each criterion component in the evaluation vector. VEGA is biased against the members in the "middle," that is, solutions that are not excellent along any criterion.

Zygote The cell produced by the fusion of the male and female gametes—the sperm and the ova: a fertilized ova; the product of the fusion of the nucleus of the ovum or ovule with the nucleus of the sperm or pollen grain.

Index

ϵ -Constraint method, 144

σ_{mating} , 178

σ_{share} , 177, 179, 210

3-objective flowshop problem, 204

a priori articulation of preference, 138, 206, 214

Aarts E, 2, 10, 13, 50, 156, 169

Ability to learn, 155

Abstraction, 8

Acquired characteristics, 23, 100, 156

Active schedule generation, 112, 118

Adaptation, 11, 19, 24, 100, 147, 153, 160

Adaptive parameterization, 57

Adaptive plan, 154

Adaptive random search, 207

Additional nondominated sorting, 16, 230

Additive utility function method, 141

Adjacent job exchange, 229

Adulbhān P, 138

Advantageous variations, 159

Allele, 25, 257

Analytical methods, 289

Anderson E J, 122

ANOVA, 58, 72

Applegate D, 121

Artificial intelligence, 2, 7

Artificial neural nets (ANN), 11, 14

Bagchi S, 6, 134, 269, 278

Bagchi Tapan P, 66, 79, 189

Baker K, 4, 5, 77, 78, 112, 120

Balas E, 13, 54, 103, 115, 121, 126, 261

Beale G O, 208

Bean J, 123, 127

Belew R K, 56

Beta distribution, 62

Bhatnagar A, 116

Bi-criteria problems, 14, 16, 203

Bi-objective maximization, 144

Bi-objective open shop, 17, 265

Biological moorings of evolution, 148

Biotic factors, 162

Birch L C, 164

Biston betularia moth, 159

Blackstone J, 120

Blazewicz J, 5

Blue tit birds, 156

Boreland Turbo C++®, 108, 291

Bouffouix S, 122, 125, 133

Box G E P, 20, 43

Bremermann H J, 20, 147

Brucker P, 12

Bruns Ralf, 57

Building block hypothesis, 51

Building blocks, 47, 51, 173

Buzacott J A, 307

Calibration, 65

Campbell H, 86, 87

Carter M, 17, 92

Caruana R A, 57

Castillo E D, 67, 72

CDS heuristic, 86, 87, 102, 287

Central dogma, 101, 149

Chakraborty C, 18

Chang Y L, 79, 90

Chen C, 96

Cheng R, 94, 102, 112, 132

Chromatid, 26

Chromosomal theory of inheritance, 153

Chromosome representation, 129

Chromosome, 20, 25, 92, 150, 200, 218

Classroom scheduling, 6, 17

Clustering, 185

Cochran W G, 57, 66, 69

Cochrane J L, 138

Coded job sequence, 156
Coding, 29, 68
Coexisting populations, 157
Combinatorial Optimization, 1, 11, 273
Combinatorial problems, 99, 112
Completion time-based GA, 127
Conflicting criteria, 138, 203
Conflicting scheduling objectives, 223
Conjunctive arc, 115
Constraint-directed approach, 121
Constraint-respecting search, 2
Context (technology)-dependent, 257
Continuous flowshop, 96
Copying error, 27
Cox G M, 57, 66, 69
Crick-Watson discovery of DNA structure, 147
Critical path methods (CPM), 3
Croce F, 123, 125
Crossover probability (p_c), 33, 56, 59
Crossover, 14, 27, 32, 96, 130, 227, 258
Crowding, 175, 187
Cultural evolution, 45, 154
CX (cycle crossover), 94

Daniels R, 6

Dannenbring D, 86, 88, 90
Dannenbring heuristic (RA), 88
Darwin C, 1, 17, 19, 158
Darwinian adaptation, 155
Darwinian and Lamarckian theories, 92, 99
Darwinian evolution, 155
Darwinian GA (also called SGA), 93, 101, 155, 287
Darwinism, 21, 22
Darwin's theory of evolution, 99, 148
Davidor Y, 94
Davis L, 57, 66, 123, 133, 179
Dawkins Richard, 18, 147
De Jong K A, 56, 185, 216
De Robertis E D P, 147
De Robertis Jr. E M F, 147
Deb K, 14, 29, 54, 137, 171, 181, 215
Defined sequence, 150
Defining length, 47
Demand-level vector, 206
Design array, 191, 196
Design noise array experiments, 192
Design of experiments, 55, 96, 200, 211

Design parameters, 195
Desired target, 192
Deterministic machine scheduling, 5
Deviation variables, 141
Differential reproduction, 166, 203
Differential reproductive performance, 161
Diploid organisms, 27
Dipoidy , 169
Disjunction graph, 115
Disjunctive constraints, 113
Disjunctive-graph-based GA, 126
Dispatching heuristic, 119
Diversification, 162
Diversity, 179, 186
Divine creation, 21
DNA (deoxyribonucleic acid), 25, 101, 147
Dobzahansky Theodosius, 167
Domain-specific chromosome, 134
Domain-specific knowledge, 156
Dorndorf U, 123, 126, 134, 261
Double helix, 150
Draper N R, 200
Dudek R, 5, 7, 86
Dummy fitness, 182, 186, 187, 212, 214

Ecker K, 5

Ecological niche, 161
Ecological theory, 159
Ecology, 148
Ecosystem, 161
Efficient front (Pareto optimal front), 142
Efficient solutions, 142, 145
Electronic filter, 193
Elite fraction (ϵ), 55
Elitism, 16
Elitist GA, 14, 217
Elitist nondominated sorting GA (ENGA), 14, 17, 217, 241, 274
Elmaraghy H A, 5
Emmons Hamilton, 14
Encoding, 7
Engineered evolution, 158
Enkawa T, 97
Enscore E, 86
Enumeration, 12, 111
Environement, 161
Environmental stimuli, 149, 160
Enzyme, 150
Equational cell division, 152

- ERP, 18
 Evolution, 1, 16, 44, 148, 154, 167
 Evolutionary computing, 288
 EVOP, 20, 43
 Exact solutions, 278
 Exploitation of good solutions, 56
 Exploration of the total search space, 55
 Expressed pattern, 163
- F**actorial experimental design, 97
 Factorial parameteric study, 65
 Falkenauer E, 123, 125, 133
 Fertilization, 152
 Fibonacci search, 59
 Fitness Function, 30
 Fitness landscape, 28, 52
 Fitness sharing, 184, 282
 Fitness, 175
 Fleming Peter J, 178, 282, 287
 Flexible manufacturing systems (FMS), 4, 5, 14, 287
 Flippone S F, 293
 Flowshop sequencing, 80, 95, 97
 Flowshops, 5, 6, 15, 80, 92
 Fonseca Carlos M, 178, 282, 284
 Food crop, 93
 Food web, 160
 Formation of species, 161
 Fox Mark S, 2, 12, 112, 122, 278
 French S, 5, 12, 83
 Fundamental niche, 163
- G**A (data-) structure, 157
 GA by Gen, Tsujimura and Kubota, 94
 GA itself used at a metalevel, 56, 96
 GA parameterization (see parameterization)
 Gametes, 25, 152
 Gantt charts, 3
 Garey M R, 80, 111
 Gelders L F, 14
 Gemmill D D, 5
 Gen M, 94, 102, 124, 132, 256
 Gene, 30, 150, 151
 Gene exchange, 168
 Gene pool crossover, 53
 Gene pool, 168
 General flowshop, 79
 General job shop, 278
- Generate-and-test method, 9
 Generation, 20, 31, 45
 Genesis, 157, 172
 Genetic algorithms for job shop, 122
 Genetic algorithms, 1, 13, 83
 Genetic divergence, 166
 Genetic drift, 158, 171, 185, 210
 Genetic information, 149
 Genetic material, 156
 Genetic plan, 147
 Genetic variations, 160
 Genetically open systems, 168
 Genome, 25
 Genotype domain, 177
 Genotype, 25, 147, 151, 156
 Germ cell, 25
 Gibbons Jean D, 247
 Giffler B, 124, 129, 133
 Glass C A, 122
 Global criterion method, 139
 Global exploration, 129
 Global left-shift, 111
 Global optima, 156
 Glover Fred W, 10, 14, 54
 Goal programming, 141
 Goals, 206
 Goldberg D E, 18, 46, 93, 172, 179, 231
 Gould Stephen Jay, 22, 24, 28, 160
 Greenberg H, 112
 Grefenstette J J, 59, 65, 94, 96, 172
 Grierson D E, 53
 Guided local search, 13
 Gupta heuristic, 87, 287
 Gupta J, 86, 87
 Gupta S, 5, 115
- H**abitat, 166
 Hajela Prabhat, 54
 Hancock P, 134
 Hans A E, 204
 Haploid, 26, 27, 280
 Hart W E, 54
 Haupt R, 120
 Heredity, 21
 Heritable differences, 160
 Heritable variations, 160
 Heuristic methods for flowshop, 92
 Heuristic methods for scheduling, 117
 Heuristic methods, 2, 157
 Heuristic search, 142, 200

Heuristics, 9, 13
Himmelblau function, 35
Historical evolution, 101, 155
Ho-Chang heuristic, 56, 79, 90, 96, 102, 104, 107, 288
Hogg G, 120
Holland J H, 11, 21, 46, 67, 93, 147, 175
Holsapple C, 123, 125, 129
Homo sapiens, 24
Homologous chromosomes, 152
Horn J, 172, 179, 180
Hundal T S, 85
Hwang C L, 138, 139
Hybrid GA approach, 96, 128, 133, 134
Hybrid method, 45, 46, 53
Hybridization, 94, 103, 168, 261
Hyperplanes, 11, 47

Ignall E, 84
Iima H, 5
Illegal offspring, 259
Immutable sequence, 78
Induced speciation, 177
Industrial quality components, 193
Inferior point, 185
Information, 32, 54
Inheritance of acquired characters, 148, 150
Inheritance, 26, 150
Injection molding, 283
In-process inventory, 82
Integer programming, 112
Intelligence, 7, 157
Interaction, 58, 64, 160, 164, 190
Inter-species competition, 163
Interspecific struggle, 158
Intraspecific competition, 164
Intraspecific struggle, 164
Inversion, 20, 28, 34
Ishibuchi H, 96, 106, 190, 232

Jain N, 108, 291, 308
Jayaram K, 84, 218, 227
Job shop, 5, 6, 77, 109, 116
Job-based GA representation, 125
Job-pair exchange mutation, 132, 260
Job-pair-relation based GA, 126
Johnson D S, 80, 111
Johnson E, 140

Johnson S, 12, 80, 82
Johnson's rule, 12, 82
Kawamura K, 6, 135, 269, 278
Keeney R, 144
Kelton 13, 69, 234, 246
Kennedy S, 94
Knight Kevin, 10
Knowledge, 7, 8, 156
Knuth's random number generator, 37
Kobayashi S, 123
Krishnakumar K, 54, 280
Kubota E, 94, 124, 129, 256
Kumar Madhu Ranjan, 189
Kumar Subodha, 57

Laguna M, 10, 13, 54
laissez-faire economics, 24, 159
Lamarckian adaptation, 149, 155
Lamarckian GA, 93, 108
Lamarckian local climb, 102
Lamarckian theory of evolution, 100, 149
Lamarckism, 24, 102, 154
Law Averill M, 13, 69, 234, 246
Learning, 10, 11, 17, 107, 154
Lenstra J K, 2, 10, 50, 80, 155, 157, 169
Lethal formation of inexpedient, 179
Lethal recombinations, 177
Lietmann G, 139
Life processes, 147
Linear programming, 114
Local adaptation, 24, 155
Local climb, 94, 102
Local enumerative search, 134
Local exploitation, 129
Local left-shift, 111
Local search heuristics, 157
Locus, 25, 27
LOX crossover, 134

Machine scheduling, 3, 5
Machine-based GA representation, 127
Magazine M J, 4, 13
Magill F N, 148, 156
Makespan, 80, 98, 221
Management objective, 136, 204
Markov chain GA models, 276
Masud A S M, 138, 140
Mathematical programming, 12, 15

- Mathias K, 226
 Mating pool, 33, 172, 226
 Mating restriction, 177, 215
 Maxwell W, 5
 MCDM, 136, 138, 283
 McKay K, 4
 Mean flow time, 82
 Mean flowtime module, 223
 Mean tardiness module, 224
 Meiosis, 26, 153
 Meiotic cell division, 150
 Mendel's factors, 21
 Messenger RNA (*m*-RNA), 152,153
 Meta-heuristic methods, 10, 13
 Meta-heuristic search, 83
 Method of distance function, 205, 206
 Method of min-max formulation, 205
 Method of objective weighting, 204
 Micro-GA, 280
 Minimize idle time heuristic, 89
 Minimum deviation method, 140
 Min-max formulation, 205
 Missed and exceeded genes, 257
 Mitchell Melanie, 46, 50
 Mitosis, 26, 151
 Mitotic, 149
 Miyabe Y, 6, 134, 269, 278
m-machine flowshop, 83
 MOGA, 179
 Monte Carlo simulation, 13
 Montgomery D C, 57, 62, 66, 72, 191, 233
 Morton T, 5, 13
 Most favored solution, 203
 Move desirability of jobs, 97
m-RNA, 152
 Muhlenbein Heinz, 46, 53, 55, 169, 280
 Multi-criteria scheduling problems, 145
 Multi-modal function, 172, 177, 178, 184
 Multiobjective decision problems, 3, 273
 Multiobjective flowshop, 16, 215, 240
 Multiobjective GA, 112, 180
 Multiobjective job shop, 14, 112, 256
 Multiobjective open shop scheduling, 14, 267
 Multi-Objective Optimization GA (MOGA), 179
 Multiobjective optimization, 136
 Multiobjective scheduling, 1, 17
 Multiple criteria optimization, 142
 Multiple-criteria decision making, 136
 Multistart approach, 10
 Murata T, 57, 96, 106, 180, 232
 Mutant, 26
 Mutation probability (p_m), 35, 36, 53
 Mutation, 1, 14, 26, 95, 132, 160, 260
 Muth J, 5
 Myers R H, 68
- N**afpliotis N, 172, 179, 180, 282
- Nakano R, 123, 126
 Natural adaptation, 60
 Natural evolution, 60, 147, 148
 Natural GA, 153
 Natural inheritance, 170
 Natural selection, 20, 100, 154, 157
 NEH heuristic, 88, 287
 Neighborhood, 13
 Neppalli R V, 96
 Neural system, 156
 Niche compacting, 163
 Niche formation, 1, 16, 28, 145, 162, 186
 Niche, 30, 161, 174
 Niching pressure, 171
 Niching, 283
 Nishikawa Y, 123, 128
 Noise array, 191
 Noise array, 191, 196
 Noise, 191, 193, 194
 Noise-induced variance, 196
 Nondelay schedule, 118
 Nondominance, 178
 Nondominant solution, 144, 275
 Nondominated front, 213
 Nondominated sorting GA (NSGA) , 14, 142, 171, 183
 Nondominated sorting, 141, 171, 181, 217
 Nondomination fitness, 291
 Nondomination, 143
 Noninferior solutions, 143
 Non-Pareto approaches, 281
 Norman B, 123, 127
 NP Hardness, 1, 10, 83, 203
 NP-complete, 80, 278
 NP-hard optimization problems, 111, 288
 Nucleic acids, 26
 Nucleotides, 150
- O**bjective conflict, 203

- Objective weighting, 17, 204
 Off-line convergence, 52, 55
 Off-line performance, 60, 76
 One-point crossover, 94, 225
 On-line convergence, 52
 On-line performance, 52, 60, 177, 214
 Open shop, 5, 6, 275
 Operation-based GA representation, 124, 257
 Order of the schema, 47
 Order statistics, 62
 Orthogonal array, 66, 187, 188
 OX (order crossover), 94, 134
- P**akath R, 123
- Palmer D, 85, 86
 Palmer heuristic, 86, 287
 Panwalkar S, 7, 120
 Paradios, 130
 Parallel (diffusion) GA, 187
 Parallel machine scheduling, 5
 Parameterization of the GA, 15, 55, 75, 97, 232, 234, 280
 Parasites, 160
 Paredis J, 123
 Pareto dominance, 280
 Pareto domination tournament, 179
 Pareto front (see efficient front), 171, 181, 280
 Pareto optimal sequences, 223
 Pareto optimal solutions, 15, 54, 141, 171, 255, 277
 Pareto optimal surface, 172
 Pareto optimality, 3, 19, 136, 143, 207, 279
 Pareto ranking, 218, 287
 Pareto Vilfredo, 144
 Pareto-optimal front, 201, 235
 Pareto-optimal regions, 184
 Partial schedule exchange crossover, 138, 259
 Partial schedule, 131
 Partially mapped crossover, 277
 Pashkevich, 278
 Patnaik L M, 56, 57
 Pentico D, 5, 13
 Performance measures, 75
 Pesch E, 123, 126, 132, 134, 261
 Phadke M S, 70, 189, 196, 197
 Phenotype sharing, 216
 Phenotype, 23, 28, 149, 160
- Phenotypic distance, 186
 Phenotypic reactions to environment, 163
 Pilot design experiments, 200
 Pilot runs, 69
 Pinedo M, 4, 6, 7, 12, 83, 92, 121, 287
 PMX (partially mapped crossover), 94, 96
 Population diversification, 163
 Population diversity, 60
 Population genetics, 154
 Population size (p_s), 55, 59
 Potts C N, 122
 Prasad Jugal, 233
 Precision desired, 67
 Predators, 159, 161
 Preference data, 140
 Preference-list-based GA, 125, 138
 Preferred solution, 142
 Premature convergence, 179
 Preselection, 187
 Priority dispatching procedures, 117
 Priority factors, 143
 Priority-rule based GA, 125
 Probability of crossover, 33, 55, 59
 Probability of mutation, 35, 36, 178
 Problem domain-specific knowledge, 54
 Processing stages, 80
 Production planning, 3
 Proportionate selection, 60
 Protein synthesis, 149
 Proteins, 26, 100, 148
 Prototype designs, 191
 Punch W F, 134
- Q**uality engineering, 187
- Quality, 199
- R**ace, 170
- Racehorses, 94
 RACS (Rapid Access with Close Order heuristic), 90, 101, 288
 RAES (Rapid Access with Extensive), 90
 Ragsdell K M, 34
 Rajendran C, 80
 Rajgopal J, 85
 Ramamoorthy C V, 96
 Raman Krishan, 76

- Random initial population, 221
 Random key-based GA, 127
 Random number generator, 37
 Randomized dispatching heuristics, 120
 Ranking selection method, 209
 Rao P Narayana, 283
 Rao S S, 196
 Rapid access (RA), 87
 Rawlings G, 46
 Realized niche, 163
 Recombination (see crossover), 27, 168
 Reddi S S, 96
 Reduction division, 152
 Reeves C, 95
 Reklaitis G V, 35
 Relative deviations, 206
 Repair methods, 94, 227
 Representation, 124, 129, 134
 Reproduction, 31, 229
 Reproductive cells, 151
 Reproductive isolation, 157, 169
 Reproductive potential, 182
 Response functions, 195
 Response surface methodology (RSM), 199
 Reward schemes, 77
 Rich Elaine, 10
 Richardson J J, 170, 174, 282
 Rinnooy Kan A, 13
 Rintalla, 187
 RNA, 26, 101, 148, 149, 282
 Robust design, 16, 66, 187, 191
 Robustness seeking factors, 192
 Robustness, 43
 Roghson J, 11, 20, 145
 Ross P, 134
 Roulette-wheel probability, 34, 38
 Roy B, 138
 Russell Peter, 18, 21, 100, 147, 169
- S**addle points, 280
 Sakawa, 144
 Salaff S, 11, 20, 145
 Sannomiya N, 5
 Sastry K K N, 18
 Satisficing, 54, 142
 Scalarized single-objective problem, 106
 Schaffer J D, 56, 185
 Scheduling, 1, 90, 122
 Schema processing, 51
 Schema theorem, 34, 48, 50, 55
 Schemas, 21, 47
 Schemata, 183
 Scudder G, 5
 Search Process, 9, 30
 Search, heuristic, 92
 Selection, 58, 146, 181, 209
 Selection/reproduction step, 38
 Selective pressure, 60
 Semi-active schedule, 110
 Sensitivity analysis, 188
 Seo Fumiko, 144
 Separability, 196, 198
 Sethi R, 80
 SGA, 30, 37, 183, 200, 203
 Shanthikumar, J George, 207
 Shared dummy fitness, 226
 Shared fitness, 186
 Sharing function, 175, 183, 187
 Sharing, 28, 174, 178, 214, 217
 Sharma Puneet, 200
 Shifting bottleneck heuristic, 102, 121, 263
 Simon H A, 141
 Simulated annealing, 10, 13, 58, 95
 Single-objective approach, 139
 Single-objective job shop, 15, 75, 112, 136
 Single point solution, 206
 Single-objective flowshop, 1, 15, 77
 Single-objective robust designs, 200
 Smith John M, 11, 99, 153, 169, 180, 186
 Solution-encoding approaches, 124
 Solution-generating heuristics, 83, 86, 107
 Solution-improvement heuristics, 83, 90
 Somatic (body) cells, 158
 Somatic variations, 159
 Sort keys, 128
 Special precedence structure, 77
 Specialization, 161, 164
 Speciation, 16, 19, 25, 146, 166, 203
 Species, 157, 174
 Srinivas M, 57
 Srinivas N, 14, 53, 136, 171, 181, 186
 Sriskandarajah C, 75
 Stahl Franklin W, 149
 Starkweather T, 224
 Static flowshop, 17
 Statistical experiments, 191
 Stochastic remainder selection, 226
 Stopping condition, 232
 Strategic combination of solution methods, 102

String length, 53, 66
Struggle for existence, 158
Suh N P, 193
Superior points, 208
Survival of the fittest, 1, 20, 159
Sussmann B, 140
Swap of two genes, 133
Symbiosis, 165

Tabu search, 10, 13, 18, 54
Tabucanon M T, 3, 140
Tackett W A, 46
Taguchi method, 191
Taillard's benchmark problems, 95
Tamaki H, 123, 126
Tanaka H, 96, 106, 180, 232
t-dom, 180
Technological constraints, 83, 109
Templeton J G C, 66, 76
Thompson G, 5, 117, 130, 133
Timetabling, 3
Transfer of inheritance, 152
Travelling salesman problem, 101,
Tripartite GA, 71
TSP, 101, 135
Tsujimura Y, 94, 112, 122, 131, 256
Two-machine flowshop, 83
Two-step Taguchi procedure, 191

Uckun S, 134, 269, 278
Utility function method, 140
Utopian solution, 173, 287

Vaccari R, 134
Van Wassenhove L, 14
Variance reduction, 71, 246
Vazacopoulos A, 13, 54, 103, 133
Vector evaluated GA (VEGA), 106, 171
Vector optimization, 185
VEGA, 106, 172, 173, 281

Vempati V, 96
Verification runs, 285
Vining G G, 66
Vose M D, 46, 52

Watson J D, 147
Weight factors, 204
Weighted sum method, 106, 180
Well behaved functions, 145
Wellman M A, 5
Whitley L D , 46, 65, 224
Workforce scheduling, 3

Yamada T, 123, 126, 127
Yamamoto N, 96
Yamamura M, 123
Young J Z , 26, 147

Zaveri J, 123
Zawack D, 121, 271
Zegrodi S H, 97
Zeleny Michael, 3, 18, 138, 279
Zweben M, 2, 12, 112, 122, 278
Zygote, 151