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

Originally published by Kluwer Academic Publishers in 1999
Softcover reprint of the hardcover 1st edition 1999

Printed on acid-free paper.

This book is dedicated to

Chhoto Ma

And to the fond memory of

Chhoto Kaku

Table of Contents

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-habitant 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-habitant 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.

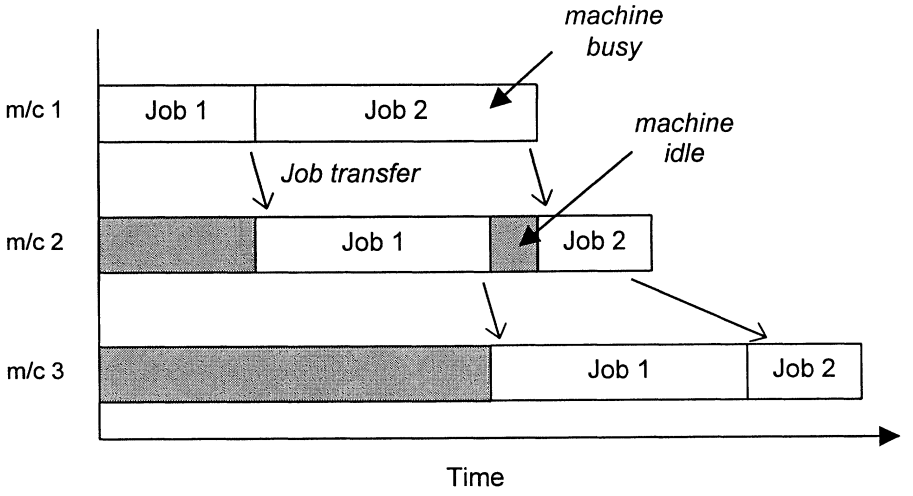

FIGURE 1.1 THE GANTT CHART

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 Kyparisis, 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 Thonpson, 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 or 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