This is to certify that the

dissertation entitled

A GENETIC ALGORITHM-BASED SCHEDULING SYSTEM FOR DYNAMIC JOB SHOP SCHEDULING PROBLEMS

presented by

Shyh-Chang Lin

has been accepted towards fulfillment
of the requirements for

Ph.D. degree in Electrical Engineering

Major professor

Date March 17, 1997

A GENETIC ALGORITHM-BASED SCHEDULING SYSTEM FOR DYNAMIC JOB SHOP SCHEDULING PROBLEMS

By

Shyh-Chang Lin

A DISSERTATION

Submitted to
Michigan State University
in partial fulfillment of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

Department of Electrical Engineering

1997

ABSTRACT

A GENETIC ALGORITHM-BASED SCHEDULING SYSTEM FOR DYNAMIC JOB SHOP SCHEDULING PROBLEMS

By

Shyh-Chang Lin

In manufacturing systems, inputs of resources, such as materials, labor, machines, energy, and information, are transformed to finished products for output. Managing the transformation process in an efficient and effective manner has been recognized as essential to survival in the current competitive marketplace. Among the operations-management functions, scheduling, which is the last step before operations plans are converted into productive activities, is concerned with allocating available resources to specific jobs and orders in the best manner to meet the operations objectives.

The goal of this research is to develop an efficient genetic algorithm-based scheduling system to address a general scheduling problem -- the dynamic job shop scheduling problem. Based on the Giffler and Thompson algorithm, we have extended that approach by providing two new operators, THX crossover and mutation, which better transmit temporal relationships in the schedule. The approach produced excellent results on standard benchmark job shop scheduling problems. We further tested many models and scales of parallel genetic algorithms in the context of job shop scheduling problems. In our experiments, the hybrid model consisting of coarse-grain GAs connected in a fine-grain-GA-style topology performed best, appearing to integrate successfully the advantages of

coarse-grain and fine-grain GAs.

In the simulation study, the objective functions examined were weighted flow time, maximum tardiness, weighted tardiness, weighted lateness, weighted number of tardy jobs, and weighted earliness plus weighted tardiness. We further tested the approach under various manufacturing environments with respect to the machine workload, imbalance of machine workload, and due date tightness. The results indicate that the approach performs well and is robust with regard to the objective function and the manufacturing environment in comparison with priority rule approaches.

To my parents

ACKNOWLEDGMENTS

First and foremost, I would like to express my deep gratitude to my advisor, Dr. Erik D. Goodman, for introducing me to Genetic Algorithms, and his invaluable help and patience. Without his guidance and expert knowledge in Genetic Algorithms, this dissertation would not have been possible. I am grateful for the opportunity to learn from his example as both a researcher and an engineer.

I would like to thank the other members of my committee, Dr. William F. Punch, Dr. Michael Shanblatt, and Dr. Gary Ragatz for many helpful comments on this research. My special thanks to Dr. Punch for giving constructive criticism in many fruitful discussions.

Thanks to the members of the GARAGe research group for their assistance and comments. Special thanks go to Dr. Min Pei and Ying Ding for many stimulating discussions.

I would also like to thank my friends and colleagues for giving help, advice, and encouragement throughout my doctoral work.

Finally, I would like to thank my parents for providing me with the opportunity to complete this study. Their love and support accompanied me through the years of this work.

TABLE OF CONTENTS

LIST OF TABLES ..... **ix**

LIST OF FIGURES ..... **xi**

1 INTRODUCTION **1**

1.1 Problem Statement 3

1.2 Goal of the Research 4

1.3 Overview of the Dissertation 6

2 BACKGROUND **8**

2.1 Genetic Algorithms 8

2.2 Job Shop Scheduling Problems 13

2.2.1 Static JSSPs 14

2.2.2 Dynamic JSSPs 22

2.2.3 Types of Schedules 22

2.3 Previous Approaches 23

2.3.1 Priority Rules 25

2.3.2 Local Search Procedures 27

2.3.3 Constraint-Based Methods 30

2.4 Genetic Algorithm Methods 32

2.5 Summary 39

3 The GA-based Scheduling System **42**

3.1 Static Model 42

3.1.1 Representation 45

3.1.2 Crossover 47

3.1.3 Mutation 49

3.1.4 Objective Functions 51

3.2 Dynamic Model 51
3.2.1 Genetic Operators 51
3.2.2 Time Decomposition Method 52
3.2.3 Rescheduling Method 54
3.2.4 Objective Functions 56

4 Parallel GA Architectures 58
4.1 Premature Convergence Problem 58
4.2 Coarse-grain GAs 61
4.2.1 Migration Method 62
4.2.2 Connection Scheme 63
4.2.3 Node Homogeneity 64
4.2.4 Injection Island GAs (iiGAs) 65
4.2.4.1 iiGA Heterogeneity 65
4.2.4.2 iiGA Migration Rules 66
4.2.4.3 iiGA Advantages 67
4.3 Fine-grain GAs 69
4.4 Hybrid PGA Models 70

5 Computational Study - Static JSSPs 72
5.1 The Effect of Parallelizing GAs 72
5.2 Comparison of PGA models 78
5.3 Other Benchmark Results 81
5.4 Summary 83

6 Computational Study - Dynamic JSSPs 86
6.1 Deterministic dynamic JSSPs 87
6.1.1 Experimental Design 87
6.1.2 Experimental Results and Discussion 89
6.2 Stochastic dynamic JSSPs 99
6.2.1 Experimental Design 99
6.2.2 Experimental Results and Discussion 100
6.3 Summary 115

7 Conclusions 117
7.1 Summary 117

7.2 Contributions 119
7.3 Future Research 121

BIBLIOGRAPHY 124

LIST OF TABLES

2.1 An example of 6x6 JSSP 18
2.2 An example of 3x3 | | Cmax 21
2.3 A list of example priority rules 26
2.4 G&T-algorithm-based GA approaches 36
3.1 Objective functions in dynamic JSSPs 57
5.1 Best Results obtained by previous approaches on the two FT problems 73
5.2 The population structures of the PGA models 78
5.3 Computational results of the ABZ benchmarks 81
5.4 Computational results of the YN benchmarks 82
5.5 Computational results of the ORB benchmarks 84
6.1 The characteristic of the test deterministic JSSPs 88
6.2 Priority rules for (normalized) weighted flow time in deterministic dynamic JSSPs 90
6.3 Priority rules for (normalized) weighted tardiness in deterministic dynamic JSSPs 91
6.4 Priority rules for (normalized) maximum tardiness in deterministic dynamic JSSPs 92
6.5 Priority rules for (normalized) weighted lateness in deterministic dynamic JSSPs 93
6.6 Priority rules for (normalized) weighted number of tardy jobs in deterministic dynamic JSSPs 94
6.7 Priority rules for (normalized) weighted earliness plus weighted tardiness in deterministic dynamic JSSPs 95

6.8 The average percentage of improvement over the best found by priority rules and Fang's GA and the corresponding level of significance α. 98
6.9 The normalized results of the GA and the percentage improvement over the priority rules 114

LIST OF FIGURES

2.1 A classical genetic algorithm 9
2.2 An example of machine Gantt chart 19
2.3 An example of job Gantt chart 19
2.4 Disjunctive graph for 3x3 | | Cmax (Table 2.2) 21
2.5 A complete selection for 3x3 | | Cmax (Table 2.2) 21
2.6 The Giffler and Thompson algorithm 24
3.1 Four types of approach for the validity problem 43
3.2 General representation of JSSPs 45
3.3 Example of representation 46
3.4 An example of THX crossover 48
3.5 An example of mutation 50
3.6 An example of time decomposition method 53
3.7 An example of modifying a schedule from the last population to the initial population of the new problem 55
4.1 Examples of the hybrid models 71
5.1 Average (100 runs) best results for three test models, various population sizes 74
5.2 Single-population GAs with popsize 100 and 250 75
5.3 Island I GA with popsize 100 and 250 76
5.4 Average best of the five PGA models with various population sizes 79
6.1 Comparison of priority rules and GA approaches for (normalized) weighted flow time in deterministic dynamic JSSPs 90
6.2 Comparison of priority rules and GA approaches for (normalized) weighted tardiness in deterministic dynamic JSSPs 91

6.3

6.4

6.5

6.6

6.7

6.8

6.9

6.1

6.1

6.1

6.1

6.1

6.1

6.1

6.3 Comparison of priority rules and GA approaches for (normalized) maximum tardiness in deterministic dynamic JSSPs 92
6.4 Comparison of priority rules and GA approaches for (normalized) weighted lateness in deterministic dynamic JSSPs 93
6.5 Comparison of priority rules and GA approaches for (normalized) weighted number of tardy jobs in deterministic dynamic JSSPs 94
6.6 Comparison of priority rules and GA approaches for (normalized) weighted earliness plus weighted tardiness in deterministic dynamic JSSPs 95
6.7 Comparison of priority rules and GA approaches for (normalized) weighted flow time in stochastic dynamic JSSPs 101
6.8 Comparison of priority rules and the GA approach for (normalized) weighted tardiness in stochastic dynamic JSSPs under balanced workload 103
6.9 Comparison of priority rules and the GA approach for (normalized) weighted tardiness in stochastic dynamic JSSPs under unbalanced workload 104
6.10 Comparison of priority rules and the GA approach for (normalized) maximum tardiness in stochastic dynamic JSSPs under balanced workload 105
6.11 Comparison of priority rules and the GA approach for (normalized) maximum tardiness in stochastic dynamic JSSPs under unbalanced workload 106
6.12 Comparison of priority rules and the GA approach for (normalized) weighted lateness in stochastic dynamic JSSPs 107
6.13 Comparison of priority rules and the GA approach for (normalized) weighted number of tardy jobs in stochastic JSSPs under balanced workload 109
6.14 Comparison of priority rules and the GA approach for (normalized) weighted number of tardy jobs in stochastic JSSPs under unbalanced workload 110
6.15 Comparison of priority rules and the GA approach for (normalized) weighted earliness plus weighted tardiness in stochastic JSSPs under balanced workload 111
6.16 Comparison of priority rules and the GA approach for (normalized) weighted earliness and weighted tardiness in stochastic JSSPs under unbalanced workload 112

CHAPTER 1

INTRODUCTION

In manufacturing systems, inputs of resources, such as materials, labor, machines, energy, and information, are transformed to finished products for output. Managing the transformation process in an efficient and effective manner has been recognized as essential to survival in the current competitive marketplace. Among the operations-management functions, operations planning seeks to determine the best utilization of available resources which will satisfy expected demand. Based on the planning horizon, there are several levels of planning [53, 54, 55]. The period of long-range planning typically exceeds 2 years. At this level, demand forecasting, facility layout and design are considered. At the middle-range planning level, the planning horizon is approximately 6 to 18 months. Aggregate planning is concerned with matching supply and demand in the medium future and setting limits on master scheduling, which is the medium term plan for production of each major product model. The short-range level is involved with scheduling, which is the last step before operations plans are converted into productive activities. While long-range planning and middle-range planning deal with the acquisition of resources, scheduling is concerned with allocating available resources to specific jobs and orders in the best manner to meet the operations objectives. The importance of scheduling is derived from the considerations of production costs and service levels. First,

bad scheduling results in poor utilization of resources. This increases production costs and reduces the competitiveness in the marketplace. Second, inefficient scheduling often delays some orders and results in unhappy customers. Therefore, scheduling of the tasks to be performed under limited resources is crucial to the efficiency and control of operations.

The construction of advance schedules to satisfy customer desires or maximize shop throughput is a very difficult and complex problem. First, there are large numbers and a variety of constraints involved. For example, there may be precedence constraints which specify a process routing of the operations for each job. Or it may not be possible to use two different resources simultaneously on the same operation. Or some resources may be unavailable during a specified period due to shifts or planned maintenance. Second, manufacturing environments in the real world are subject to many sources of change and uncertainty, such as new job releases, job cancellations, machine breakdowns, due date changes, etc. Third, scheduling typically seeks to achieve several conflicting objectives, such as low work-in-process inventories, high shop utilization, and satisfaction of customer demand. From the computational viewpoint, most scheduling problems are NP-hard [1] -- *i.e.*, the time required to compute an optimal schedule increases exponentially with the size of the problem. Because brute-force or undirected search methods are computationally prohibitive, at least for problems of any size, scheduling problems tend to be solved using a combination of search and heuristics to get optimal or near optimal solutions. Among various search methodologies used for scheduling problems, the Genetic Algorithm (GA) [9, 30], inspired by the process of Darwinian evolution, has been recognized as a general search strategy and optimization method which is often useful in

attacking combinatorial problems. In contrast to other local search techniques such as simulated annealing and tabu search, which are based on manipulating one feasible solution, the GA utilizes a population of solutions in its search, giving it more resistance to premature convergence on local minima. Since Davis proposed the first GA-based technique to solve scheduling problems in 1985 [31], GAs have been used with increasing frequency to address scheduling problems. This dissertation considers a general scheduling problem -- the job shop scheduling problem (JSSP) --, and develops a GA-based scheduling system to address it.

1.1 Problem Statement

Job shop scheduling, in general, consists of a set of concurrent and conflicting goals to be satisfied using a finite set of machines. Each job has a processing order through the machines which specifies the precedence restrictions. The importance of job j relative to the other jobs in the system is denoted by the weight w_j. The main constraint of jobs and machines is that one machine can process only one operation at a time and preemption of any operation on any machine is prohibited. Additionally, we assume that the processing times are known when jobs arrive at the shop and the machines are always available (whenever not in use by another job). Usually we denote the general JSSP as JxM, where J is the number of jobs and M is the number of machines. Based on the release times of jobs, JSSPs can be classified as static or dynamic scheduling. In *static* JSSPs, all jobs are ready to start at time zero. In *dynamic* JSSPs, job release times are not fixed at a single point; that is, jobs arrive at various times. Dynamic JSSPs can be further classified as deterministic or stochastic based on the manner of specification of the job release times.

Deterministic JSSPs assume that the job release times are known in advance. In *stochastic* JSSPs, job release times are random variables described by a known probability distribution.

In this dissertation, we mainly consider the dynamic JSSP with jobs arriving continually. Both deterministic and stochastic models of the dynamic problem are investigated. The objective functions, that is, the measure of performance, examined are weighted flow time, maximum tardiness, weighted tardiness, weighted lateness, weighted number of tardy jobs, and weighted earliness plus weighted tardiness.

1.2 Goal of the Research

The goal of this research is to develop an efficient GA-based scheduling system to address JSSPs, especially dynamic JSSPs. To achieve this goal, the following plan is organized in a stepwise and overlapping fashion.

(1) Design of GA-based Scheduling System for Static JSSPs

The main difficulty in applying GAs to highly constrained and combinatorial optimization problems such as JSSPs is maintaining the validity of the solutions (Section 3.1). This problem is typically solved by modifying the breeding operators or providing penalties on infeasible solutions in the fitness function. The issues of validity and representation are discussed. Two new operators, namely, THX crossover and mutation, which better transmit temporal relationships in the schedule, are developed.

(2) Validation of the Static GA-Based Scheduling System

Much previous research has been done on the static JSSPs and therefore many

benchmarks exist. The designed GA-based scheduling system in (1) is tested on some standard benchmark JSSPs and compared with other approaches proposed. Based on the results, possible modifications of the scheduling system are made in order to improve the performance.

(3) Design of GA-based Scheduling System for Dynamic JSSPs

The scheduling system is extended to address dynamic JSSPs. When faced with a relatively volatile environment, the scheduling system should be able to reactively revise schedules in response to unexpected events. Instead of putting a large effort into finding the optimal schedule, producing a good schedule and maintaining this performance in a dynamic environment are the main concerns. First, the scheduling system is modified to deal with deterministic JSSPs. The stochastic JSSP then is decomposed into a series of deterministic problems by the method proposed by Raman *et al.* [56]. A deterministic problem is generated at each occurrence of a nondeterministic event, and then solved by the GA. The issue of the rescheduling problem is discussed and an innovative rescheduling method, which modifies the adapted population into a new population between successive events, is proposed.

(4) Comparison of Parallel GA Architectures

Although "classical" GAs can be made somewhat resistant to premature convergence (*i.e.*, inability to search beyond local minima), there are methods which can be used to make GAs even more resistant. One approach to reduce the premature convergence of a GA is parallelization of the GA into disjoint subpopulations, which is also a more realistic model of nature than a single population.

Parallel GAs (PGAs) retard premature convergence by maintaining multiple, somewhat separated subpopulations which may be allowed to evolve more independently (or, more precisely, by employing non-panmictic mating). Various models and scales of PGAs are tested and compared in the context of static JSSPs.

(5) Computational Study of Deterministic JSSPs

A test set of 12 deterministic problems taken from [53] is tested. The scheduling system with the best PGA architecture found in (4) is compared with priority rules and another GA-based approach proposed by Fang [59] to assess the relative performance.

(6) Computational Study of Stochastic JSSPs

A simulation model of a stochastic job shop, tested in various manufacturing environments with respect to the machine workload, imbalance of machine workload, and due date tightness, is presented. The simulation model is used to provide the performance of the scheduling system for comparison with priority rules. The results are also used to assess the robustness of the scheduling system with regard to the objective function and the manufacturing environment.

1.3 Overview of the Dissertation

The rest of the dissertation is organized as follows. Chapter 2 introduces GAs and discusses the JSSPs. A review of related research on the JSSPs, especially using GA-based approaches, is presented. Chapter 3 first describes the basic framework of the GA-based scheduling system for static JSSPs. The extension of the basic framework to dynamic

JSSPs is then presented. Chapter 4 describes the models of PGA architectures. Two fundamental models, coarse-grain GAs and fine-grain GAs, and two hybrid models are presented. Chapter 5 applies the basic framework of the scheduling system described in Chapter 3 to some benchmark JSSPs. Different models of PGAs described in Chapter 4 are tested and compared in the context of JSSPs. Chapter 6 applies the extension of the basic framework to some test problems of deterministic dynamic JSSPs. The results are compared with another GA-based scheduling system and some priority rules. In stochastic JSSPs, the simulation model is described and the results are compared with some priority rules. Finally, Chapter 7 provides conclusions and suggestions for future research.

CHAPTER 2

BACKGROUND

It is well known that the JSSP is not only NP-hard [60] it is also among the hardest combinatorial optimization problems. Due to the complexity and the vast search space of JSSPs, conventional optimization methods, such as mathematical programming, dynamic programming, and branch-and-bound, are computationally infeasible. This has led to recent interest in using heuristic search methods to find a near optimal solution within reasonable computation time. Among the heuristic search methods, GAs are increasingly being used to address the JSSPs. In this chapter, a brief review of the GA is presented in the first section. The description of the JSSP studied is given next. A literature review of previous related research, especially using GA-based approaches, is covered in the last section.

2.1 Genetic Algorithms

In 1975, Holland described a methodology, later called GA, for studying natural adaptive systems and designing artificial adaptive systems [9]. The GA is now frequently used as an optimization method, based on analogy to the process of evolution. [9, 30, 61] contain a theoretical analysis of the GA. Roughly speaking, the GA is one kind of search strategy which combines survival of the fittest among the space of string structures with a

Step 1:
Randomly generate the initial population, C.
Step 2:
Evaluate fitness of all initial individuals.
Step 3:
Let N be a null population.
Step 4:
(a) Select two individuals from C with probability P_s proportional to their fitness.
(b) Recombine the two selected parents with probability P_c to form two offspring.
(c) Mutate each bit of the two offspring with probability P_m.
(d) Insert the two offspring into N.
Step 5:
Return to *Step 4* until N is full.
Step 6:
Replace C with N;
Evaluate fitness of all individuals in C.
Step 7:
Return to *Step 3* until the termination criterion holds.
Step 8:
Print the results and exit.

Figure 2.1 A classical genetic algorithm

stochastic information exchange mechanism. As opposed to many other optimization methods, GAs work with a population of candidate solutions instead of just a single solution. GAs assign a value to each string in the population according to a problem-specific objective function. A "survival-of-the-fittest" step selects strings from the old population. A reproduction step applies operators such as crossover or mutation to these strings to produce a new population that is more fit than the previous one. An informal description of a classical GA is shown in Figure 2.1. The basic components of the classical GA and some variations are explained below.

(1) Initialization

In Holland's original GAs, the encoded individual -- *i.e.*, chromosome, represents a set of binary parameters. The initialization of a population is performed by randomly generating a binary string. To reduce the sampling error, *superuniform initialization*, which guarantees that the initial population contains all possible combinations of length less than or equal log_2(population size), is developed. For non-binary representations, the initialization procedure typically randomly generates valid chromosomes such as a feasible tour in traveling salesperson problems (TSPs) and a feasible schedule in JSSPs. The main consideration of initialization is to ensure a high degree of diversity in the initial population.

(2) Fitness scaling

For a "raw" fitness-based selection mechanism, extraordinary individuals can quickly dominate the population, and similar objective function values of individuals in the population in the later phase of a GA run can rapidly reduce the selection pressure. Both situations lead to a premature stagnation of the search

process. To control the selection pressure, the objective function values are often transformed to fitness values by some scaling function. A variety of scaling methods have been proposed, such as *linear scaling* [30], *linear dynamic scaling* [62], *window scaling* [52], *logarithmic scaling* [62], *exponential scaling* [62], *Boltzmann scaling* [63], and *sigma truncation* [64]. The one used in the dissertation is sigma truncation, which is formulated as

$$fitness_i = \max(0, f_i - (f_{avg} - s*sigma)) \tag{2.1}$$

where f_i is the objective function value of individual i; f_{avg} is the average objective function value of current population; and *sigma* is the standard deviation. The parameter s is used to control the bias towards highly fit individuals.

(3) Selection

The selection mechanism assigns each individual a selection probability, P_s, then selects parents based on their P_s. The original GAs proposed by Holland use *proportional selection* (*roulette-wheel selection*) in which P_s is directly proportional to its fitness. Another fitness-based selection is *stochastic universal sampling* [65]. In contrast to proportional selection, n equally spaced pointers (instead of one pointer) are placed on a spinning wheel and a single spin results in a complete pool of mating parents. This selection mechanism has minimum spread about the desired distribution. Another selection scheme is based on the ranking of individuals according to their objective functions. One example is *linear ranking* proposed by Baker [66]. The selection probability of individual i is defined as

$$P_s(i) = \frac{1}{n}\left(\eta_{max} - (\eta_{max} - \eta_{min})\frac{i-1}{n-1}\right) \tag{2.2}$$

where η_{max} and η_{min} are the expected numbers of individuals with rank 1 and rank n respectively. Due to the constraints, the relation $1 \leq \eta_{max} \leq 2$ and $\eta_{min} = 2.0 - \eta_{max}$ must be satisfied. Brindle described a different selection scheme, *tournament selection* [67], which randomly chooses k individuals from the population and returns the fittest one as a parent.

(4) Recombination

The crossover operator is the most important search operator in GAs. The basic idea of crossover is that "useful characteristics" of different parents may be combined to produce better offspring. The traditional *one-point crossover* introduced by Holland exchanges substrings between parents starting at a point chosen randomly on the strings. Some extensions are *2-point crossover*, *k-point crossover*, and *uniform crossover* [68]. For permutation-type problems such as TSPs, the "useful characteristics" are the ordering information or the edge information. Several permutation-type crossover operators have been developed, such as *partially matched crossover* (PMX) [70], *order crossover* (OX) [72], *linear order crossover* (LOX) [88], *position-based crossover* (PX) [41], *cycle crossover* (CX) [72], *uniform order-based crossover* [69], and *edge recombination* [71].

(5) Mutation

The main purpose of mutation is to reintroduce lost alleles into the population. In the original bit mutation, each bit of the chromosome is given a probability P_m of undergoing mutation. One problem of applying the bit mutation operator in permutation-type problems is that the ordering information has nothing to do with a single bit or a single field. Many other types of mutation, such as *swap mutation*

[34] and *scramble sublist mutation* [69], can be employed on permutation-type problems.

(6) Replacement

Depending on the replacement strategy, GAs are classified as *generational* or *steady-state* [73]. The original GA proposed by Holland is generational, *i.e.*, the next generation is calculated entirely from the current generation and the next generation replaces the current generation only after the new population is completely created. In steady-state GAs, the reproduced offspring is immediately put back into the current population so the parents and offspring can co-exist. Another important replacement strategy is *elitism*, which guarantees the survival of the best individual under selection and reproduction.

(7) Termination criterion

The termination criterion is often the maximum number of generations or the maximum number of function evaluations. Other criteria used are related to diversity measures. One example is the *maximum bias* [52], in which the bias measure of a population -- *i.e.*, the percent converged loci, indicates the degree of genotype diversity. Other examples are *maximum number of converged loci* [52], *maximum number of generations passed without creating a new genome* [52], and *percent resemblance to best individual* [63].

2.2 Job Shop Scheduling Problems

The JSSP has been an active research area, partly because of the many possible applications, and also because of the great difficulty of the problem, which challenges

existing search methodologies. Roughly speaking, a JSSP may be defined as a problem of allocating resources subject to precedence and resource constraints so that some measure of performance achieves its optimal value. Based on the release times of jobs, JSSPs are categorized as *static* or *dynamic* scheduling problems [53, 59, 74]. In the following subsections, both models are described.

2.2.1 Static JSSPs

In JSSPs, the resources are called machines, and basic tasks are called jobs. Each job is a request for the scheduling of a set of operations according to a process plan (or referred to as process routing) which specifies the precedence restrictions. We adopt the three-field classification, $\alpha \mid \beta \mid \gamma$, to characterize the structure of the scheduling problem [3, 10]. The number of machines is denoted by M and the number of jobs is denoted by J. The first field, α, describes the machine environments. Because only JSSPs are considered in this dissertation, JxM is used to indicate the size of the job shop. The second field, β, indicates the number of jobs and machine characteristics, which are defined as follow,

(1) P : There are identical parallel machines in the problem, *i.e.*, some machines in parallel have equal speeds.

(2) Q : There are uniform parallel machines in the problem, *i.e.*, some machines in parallel have different speeds, which are independent of the jobs.

(3) R : There are unrelated parallel machines in the problem, *i.e.*, some machines have different speeds, which are dependent on the jobs.

(4) a : Each job has one or more alternative process plans.

(5) b : Machines break down unexpectedly or are unavailable due to scheduled maintenance. If P, Q, or R is present, some jobs can be rescheduled.

(6) d : Jobs arrive randomly over a period of time and a job can be canceled.

(7) r : The release dates may differ for each job.

(8) s : The setup times are sequence-dependent.

The third field, γ, refers to the optimality criterion, *i.e.*, the objective to be minimized. The completion time and due date of job j are denoted by C_j and d_j, respectively. The lateness of job j is defined as

$$L_j = C_j - d_j \tag{2.3}$$

The tardiness of job j is defined as

$$T_j = max(L_j, 0) \tag{2.4}$$

The earliness of job j is defined as

$$E_j = max(-L_j, 0) \tag{2.5}$$

The unit penalty of job j is defined as

$$U_j = \begin{cases} 1 & \text{if } C_j > d_j \\ 0 & otherwise \end{cases} \tag{2.6}$$

The following are some common objective functions,

(1) C_{max} : Makespan, defined as $max(C_1, \ldots, C_n)$.

(2) T_{max} : Maximum tardiness, defined as $max(T_1, \ldots, T_n)$.

(3) $\sum C_j$: Flow time.

(4) $\sum w_j C_j$: Weighted flow time.

(5) $\sum w_j T_j$: Weighted tardiness.

(6) $\sum w_j L_j$: Weighted lateness.

(7) $\sum U_j$: Number of tardy jobs.

(8) $\sum w_j U_j$: Weighted number of tardy jobs.

(9) $\sum w_j(E_j + T_j)$: Weighted earliness plus weighted tardiness or just-in-time.

Except for the objective (1), (3), and (4), these objective functions are *due-date-related.* Also note that the weighted earliness plus weighted tardiness is a *nonregular* objective function -- finishing a job earlier may not represent improved performance.

In a classical static JSSP, $JxM| \mid \gamma$, the following are assumed [2, 5, 6].

(A1) One machine can process only one operation at a time.

(A2) Once processing begins on a job, it must be performed to completion.

(A3) Once processing begins on an operation, it must be performed to completion.

(A4) Each job is an entity; that is, no two operations of the same job may be processed simultaneously.

(A5) The setup time is sequence-independent and can be included in processing time.

(A6) The transportation time between machines is negligible.

(A7) In-process inventory is allowable.

(A8) There is only one of each type of machine.

(A9) The number of jobs is known and fixed.

(A10) The processing times are known and fixed.

(A11) Due dates, if they exist, are fixed.

(A12) All jobs are ready to start at time zero.

(A13) Machines never break down and are always available.

(A14) The process plan for each job is given and no alternative plans are permitted.

The determination of an optimal schedule can be formulated as an integer programming problem [6]. Let x_{ik} denote the completion time of job i on machine k, t_{ijk} denote the processing time of operation j of job i on machine k. To build a schedule, we have to decide all variables x_{ik}. In order to represent the precedence constraints, suppose that operation j of job i requires machine k and operation $(j\text{-}1)$ of job i requires machine h. Then in order for a set of x_{ik} to be feasible, it's necessary to have

$$x_{ik} - t_{ijk} \geq x_{ih} \qquad 2 \leq j \leq m,\ 1 \leq i \leq n \tag{2.7}$$

For the first operation $(j=1)$ the constraint is

$$x_{ik} - t_{i1k} \geq 0 \qquad 1 \leq i \leq n \tag{2.8}$$

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 that job i precedes job p on machine k. Then it's necessary to have

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

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

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

In order to accommodate these constraints in the formulation, an indicator variable y_{ipk} is defined as

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

Then the constraints become

$$x_{pk} - x_{ik} + H(1 - y_{ipk}) \geq t_{pqk} \tag{2.9}$$

$$x_{ik} - x_{pk} + H y_{ipk} \geq t_{ijk} \tag{2.10}$$

where H is a very large positive number. For the $JxM \mid\mid \gamma$ problem the entire formulation is

Minimize γ (2.11)

Subject to (2.7), (2.8), (2.9), and (2.10).

One of the oldest, simplest, and still one of the most useful methods of representing schedule information is the Gantt chart. The horizontal axis represents time. In the standard machine Gantt chart, the vertical axis indicates machines, and the jobs shown occupy the machine at each point of time. In the standard *job* Gantt chart, the vertical axis indicates jobs, and the machines shown occupy the job at each point of time. Table 2.1 shows an example of 6x6 benchmark JSSP, FT6x6 [8]. In this example, job 1 must go to

Table 2.1 An example of 6x6 JSSP

Job	Machine Routing (Processing Time)					
1	3 (1)	1 (3)	2 (6)	4 (7)	6 (3)	5 (6)
2	2 (8)	3 (5)	5 (10)	6 (10)	1 (10)	4 (4)
3	3 (5)	4 (4)	6 (8)	1 (9)	2 (1)	5 (7)
4	2 (5)	1 (5)	3 (5)	4 (3)	5 (8)	6 (9)
5	3 (9)	2 (3)	5 (5)	6 (4)	1 (3)	4 (1)
6	2 (3)	4 (3)	6 (9)	1 (10)	5 (4)	3 (1)

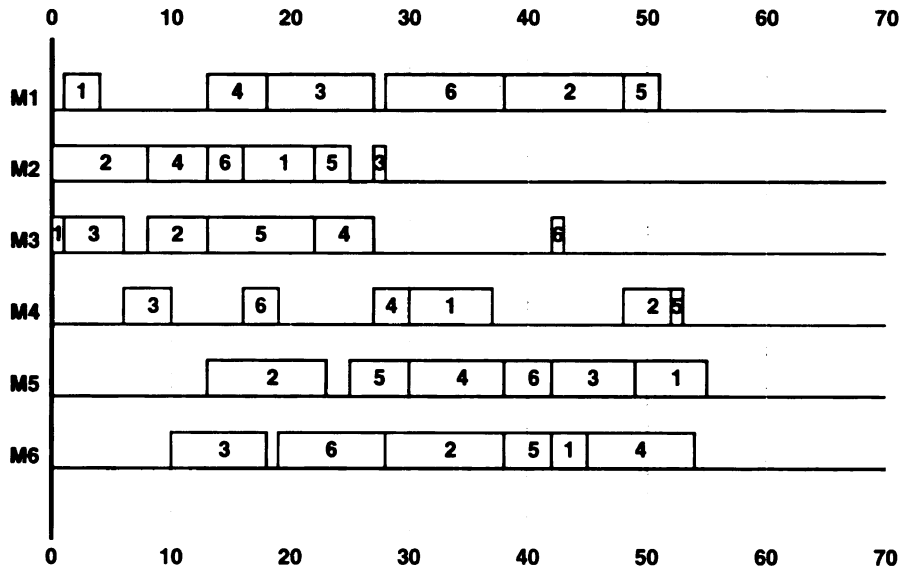


Figure 2.2. An example of machine Gantt chart

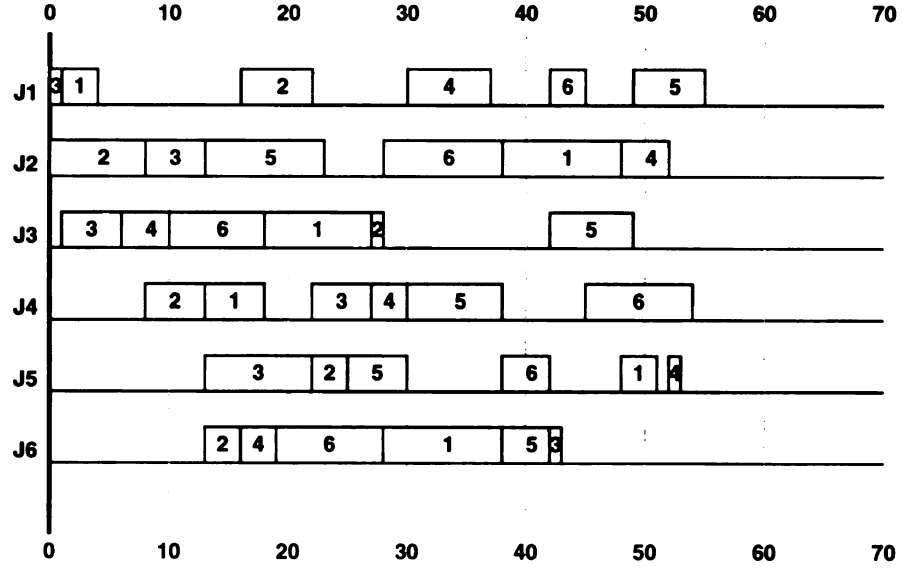


Figure 2.3 An example of job Gantt chart

machine 3 for 1 unit of time, then machine 1 for 3 units of time, machine 2 for 6 units of time, and so on. Figure 2.2 represents a *machine* Gantt chart of a feasible schedule of FT6x6 problem. Machine 1 first processes the operation of job 1 from time 1 to time 4, then the operation of job 4 from time 13 to time 18, and so on. Figure 2.3 represents a job Gantt chart of a feasible schedule of FT6x6 problem. Job 1 is first scheduled on machine 3 from time 0 to time 1, then machine 1 from time 1 to time 4, and so on. Both schedules are feasible because there are never two jobs on a machine at one time; a job is never scheduled on more than one machine at one time; and the operations of each job are processed in the order specified in Table 2.1.

The problem $JxM \mid\mid C_{max}$ can also be expressed by a disjunctive graph, $G=(N, A, E)$, where N is the node set, A is the conjunctive arc set, and E is the disjunctive arc set. The nodes N correspond to all of the operations and two dummy nodes, a source and a sink. The conjunctive arcs A represent the precedence relationships between the operations of a single job. The disjunctive arcs E represent all pairs of operations to be performed on the same machine. The length of each arc emanating from a node is the processing time of the operation performed at that node. The source has conjunctive arcs with length zero emanating to all the first operations of the jobs and the sink has the conjunctive arcs coming from all the last operations of the jobs. A feasible schedule corresponds to a selection of exactly one arc from each disjunctive arc pair such that the resulting directed graph is acyclic. The problem of minimizing the makespan (p. 15) reduces to finding a set of disjunctive arcs that minimize the length of the longest path or the critical path in the directed graph. Table 2.2 shows an example of 3x3 $\mid\mid C_{max}$. The disjunctive graph is shown in Figure 2.4. The operation of job j on machine m is denoted by operation (j, m).

Table 2.2 An example of 3x3 | | C_{max}

Job	Machine Routing (Processing Time)		
1	3 (1)	1 (3)	2 (6)
2	2 (8)	3 (5)	1 (4)
3	3 (5)	2 (4)	1 (8)

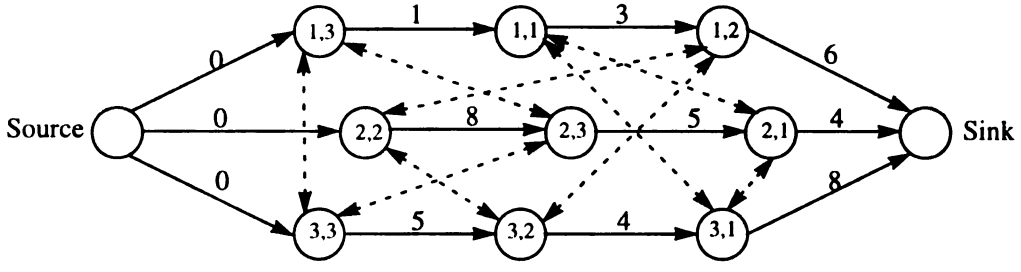


Figure 2.4. Disjunctive graph for 3x3 | | C_{max} (Table 2.2)

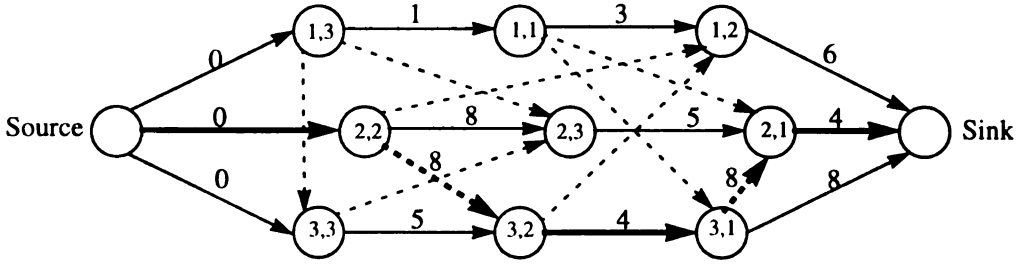


Figure 2.5 A complete selection for 3x3 | | C_{max} (Table 2.2)

Figure 2.5 shows a complete selection of the disjunctive graph. The directed graph is acyclic and the length of the critical path, in bold face arcs, is 24.

2.2.2 Dynamic JSSPs

By relaxing assumption A9 and A12 , JSSPs become dynamic. In dynamic JSSPs, job release times are not fixed at a single point, that is, jobs arrive at various times. Some examples are $JxM \mid r \mid T_{max}$ and $JxM \mid d \mid \sum w_j U_j$. Dynamic JSSPs can be further classified as deterministic or stochastic based on the manner of specification of the job release times. *Deterministic* JSSPs assume that the job release times are known in advance. In *stochastic* JSSPs, job release times are random variables described by a known probability distribution. The release time of job j is denoted by r_j. Some objective functions in last subsection are modified to take account of the job release times as follows:

(1) $\sum(C_j - r_j)$: Flow time.

(2) $\sum w_j(C_j - r_j)$: Weighted flow time.

In dynamic JSSPs, minimizing makespan is of less interest because the scheduling horizon is open and the makespan gives no credit for jobs that finish well before the last one finishes. Reducing turnaround time through the shop or reducing the amount of tardiness is usually the primary objective.

2.2.3 Types of Schedules

Because arbitrary amounts of idle time can be inserted between adjacent pairs of operations on any machine, the number of feasible schedules is infinite. It should be clear that such excess idle time is useless under any regular measure of performance. Therefore, three types of schedules without excess idle time are defined as follows.

Def

Defi

Defi

C

the sc

optim

there

approa

2.6 is

inequa

compe

any op

schedu

2.3 P

Beca

all availa

significan

Definition 2.1 A *nondelay* schedule is a feasible schedule in which no machine is kept idle at a time when it could begin processing some operation.

Definition 2.2 A *semi-active* schedule is a feasible schedule in which no operation can be completed earlier without changing the job sequence on any of the machines.

Definition 2.3 An *active* schedule is a feasible schedule in which no operation can be completed earlier by changing the processing sequence on any of the machines without delaying some other operation.

Clearly the set of active schedules is a subset of the set of semi-active schedules, and the set of nondelay schedules is a subset of the set of active schedules. Furthermore, in optimizing regular objective functions, the optimal schedules are active schedules but there is no guarantee that the nondelay subset contains an optimum. A systematic approach to generate active schedules was proposed by Giffler and Thompson [46]. Figure 2.6 is a brief outline of the G&T algorithm. The key condition in the G&T algorithm is the inequality $r_{jm^*} < t(C)$ in Step 3, which generates a conflict set consisting only of operations competing for the same machine. Once one operation is decided, it is impossible to add any operation that will complete prior to $t(C)$, making the generated schedule an active schedule.

2.3 Previous Approaches

Because a considerable amount of literature has been created, a thorough discussion of all available approaches would be impossible. We will restrict ourselves to the most significant heuristic approaches, paying special attention to recent developments and

Step 1:
Let $\boldsymbol{C}$ contain the first schedulable operation of each job;
Let $r_{jm} = 0$, for all operations (j, m) in $\boldsymbol{C}$.
(r_{jm} is the earliest time at which operation (j, m) can start.)

Step 2:
Compute $t(\boldsymbol{C}) = \min_{(j,m) \in \boldsymbol{C}} \{r_{jm} + p_{jm}\}$
and let m^* denote the machine on which the minimum is achieved.

Step 3:
Let $\boldsymbol{G}$ denote the conflict set of all operations (j, m^*) on machine m^* such that $r_{jm^*} < t(\boldsymbol{C})$.

Step 4:
Randomly select one operation from $\boldsymbol{G}$ and schedule it.

Step 5:
Delete the operation from $\boldsymbol{C}$;
include its immediate successor in $\boldsymbol{C}$;
update r_{jm} in $\boldsymbol{C}$ and return to step 2 until all operations are scheduled.

Figure 2.6 The Giffler and Thompson algorithm

omitting the details. A more comprehensive survey can be found in [2, 3, 11, 12, 13, 14, 15].

2.3.1 Priority Rules

The most often used heuristic method is the priority rule, which is also known as dispatching rule or scheduling rule. The priority is based on some easily computed parameters of the jobs, operations, or machines, such as processing times, due dates, release times, and machine loadings. Whenever a machine becomes available, the priority rule is applied and then the job with the highest priority is selected for work. A priority rule is thus dynamic in nature and continually adjusts to changing conditions. Some survey papers of priority rules can be found in [16, 17, 75, 76, 77]. Priority rules are computationally efficient and are useful for finding a reasonably good schedule with regard to a single objective such as the makespan, the flow time, or the maximum tardiness. However, the objectives are often more complicated in practice. Although more elaborate priority rules [17], which are basically a combination of a number of the elementary priority rules, can address more complicated objective functions, prioritizing the jobs based on only a few job or machine parameters restricts the search capability in real-world scheduling problems.

Table 2.3 is a list of the priority rules used for comparison in this dissertation. The total processing time and remaining processing time of job j are denoted as P_j and R_j respectively. Further, p_{jm} is the processing time of operation (j, m); n_j is the number of remaining operations of job j; and $\bar{p}_m$ is the average operation processing time on machine m.

Table 2.3 A list of example priority rules

Rule	Description	Priority of operation (j, m) at time t
RANDOM	Randomly select a schedulable operation	equal priority
FCFS	First Come First Serve	$1 / r_j$
WSPT	Weighted Shortest Processing Time	w_j / p_{jm}
WLWKR	Weighted Least Work Remaining	w_j / R_j
WTWORK	Weighted Total Work	w_j / P_j
EGD	Earliest Global Due date	$1 / d_j$
EOD	Earliest Operational Due date	$1 / [r_j + (d_j - r_j) R_j / P_j]$
EMOD	Earliest Modified Operational Due date	$1 / max(r_j + (d_j - r_j) R_j / P_j , t + p_{jm})$
MST	Minimum Slack Time	$-(d_j - R_j - t)$
WS/OP	Weighted Slack per OPeration	$w_j[1 - (d_j - R_j - t) / n_j)] / p_{jm}$
WCR	Weighted Critical Ratio	$w_j[1 - (d_j - t) / R_j)] / p_{jm}$
WCOVERT	Weighted COVERT	$w_j[1 - (d_j - R_j - t)^+ / 2R_j)]^+ / p_{jm}$
WR&M	Weighted R&M	$w_j \, exp[-(d_j - R_j - t)^+ / 2\overline{P}_m)] / p_{jm}$

2.3.2 Local Search Procedures

The local search procedures usually attempt to find a good schedule through a search in the neighborhood of the current schedule. A comprehensive survey of recent local search procedures can be found in [78, 79]. Basically, this approach can be specified by three components: the method of obtaining the initial schedule, called the seed, the mechanism of generating the neighborhood of the seed, and the method of selecting a schedule to replace the seed. The seed is usually generated by priority rules or other heuristic methods. The neighborhood generating mechanism is an important aspect of local search procedures. Some simple generating mechanisms are adjacent pairwise interchange and pairwise interchange. More sophisticated neighborhoods can be obtained by interchanging a pair of adjacent operations on the critical path of the directed graph which specifies the current schedule by completely selecting the disjunctive arcs. Other generating mechanisms are one-step look-back interchange, multi-step look-back interchange, one-step look-ahead interchange, and multi-step look-ahead interchange [10], which generate the neighborhood by backtracking one or more steps after an adjacent pairwise interchange is made.

The third component is the selecting method, which decides the behavior of the search procedures. Here we describe three well-known procedures.

(1) Hill Climbing

Hill climbing is a standard local search procedure. It selects the first or the best improvement in the neighborhood to replace the seed [6]. If the neighborhood of a schedule is small, then it's reasonable and feasible to generate them all and select the best. If the neighborhood is very large, then to even generate them all might take a very long

time. In this case it may be best to select the first improvement. One possible variation is selecting the best improvement from a fixed number of sampled schedules in the neighborhood. It is worthwhile noticing that there is no way to escape the first encountered local optimum since any move in the local neighborhood produces a worse schedule.

(2) Tabu Search

Tabu search [18, 20] allows the possibility that the selected schedule is worse than the current schedule in an organized way. It maintains a list of tabu restrictions to prevent the reversal, or sometimes repetition, of certain moves. Every time a move is made, the reverse move is put at the top of the list and the restriction at the bottom is deleted. If a candidate move is tabu, the aspiration criteria are given an opportunity to override the tabu status. Thus the primary goal of the tabu restrictions is to prevent the search from becoming trapped at local optima. An important consideration in tabu search is the tabu list size. If the number of restrictions in the tabu list is too small, cycling may occur. If the number of restrictions is too large, the search may be constrained improperly. The preferred tabu list sizes should lie in intervals related to problem dimension.

(3) Simulated Annealing

Unlike tabu search, simulated annealing [19] escapes from local minima in a probabilistic way. Besides accepting better schedules, it also to a limited extent accepts worse schedules to replace the seed. Let S_k and S denote the current schedule and the candidate schedule selected from the neighborhood of S_k. Also $\gamma(S_k)$ and $\gamma(S)$ denote the corresponding values of the objective function. Then the acceptance criterion determines whether S replaces S_k by applying the following acceptance probability:

$$P_{c_k}(\text{accept } S) = \begin{cases} 1 & \text{if } \gamma(S) \le \gamma(S_k) \\ \exp\left(\dfrac{\gamma(S_k) - \gamma(S)}{c_k}\right) & \text{if } \gamma(S) > \gamma(S_k) \end{cases} \tag{2.12}$$

The $c_0 \ge c_1 \ge \ldots \ge 0$ are cooling parameters which determine the speed of convergence. Because c_k decreases with k, the acceptance probability for a nonimproving move is lower in later iterations of the search process. Also, if the candidate schedule is significantly worse, the acceptance probability is very low and the move is not likely to be made.

One advantage of these three local search procedures is that they don't have to know much about the structural properties of the problem, such as derivatives of the objective function. This feature is attractive for problems in which derivatives of the objective function are unknown. If the derivative of the objective function can be determined, the advantage vanishes and these three procedures typically become inferior to gradient-type methods, since they require much computing time to evaluate each new schedule during a local search. Here we describe one gradient-type method, Hopfield neural networks (Hp NNs), which has been applied to JSSPs [21, 22, 23].

A Hp NN is characterized as a highly interconnected network of simple analog processors. The network energy decreases continuously in time and the network converges to one of the stable minima in the state space. To solve problems using NNs, the network energy function is mapped to a certain objective function that needs to be minimized. Hp NNs can solve the integer programming problem described in (2.11) [24]. The energy function is defined as

$$E = \sum_{i=1}^{n} x_{ik_i} + \sum_{i=1}^{n}\sum_{j=2}^{m} H_1 \times F(x_{ik} - x_{ih} - t_{ijk}) + \sum_{i=1}^{n} H_2 \times F(x_{ik} - t_{i1k}) \tag{2.13}$$

$$+ \sum_{k=1}^{m}\sum_{i,p} H_3 \times (F(x_{pk} - x_{ik} + H(1 - y_{ipk}) - t_{pqk}) + F(x_{ik} - x_{pk} + Hy_{ipk} - t_{ijk}))$$

where H_1, H_2, and H_3 are large positive constants. $\boldsymbol{F}$ is a nonlinear function whose derivative $\boldsymbol{f}$ is defined as

$$f(x) = \begin{cases} 0 & x \geq 0 \\ x & otherwise \end{cases} \tag{2.14}$$

Each local minimum is an attractor in the state space. The set of initial states which initiates the evolution terminating in one attractor is called the basin of attraction. The attractor and the basin of each attractor are determined by the energy function and the internal parameters of the NN. A Hp NN is a deterministic local search procedure. Once the initial state is selected, it will converge to the minimum in whose basin the initial state is located. The advantage of Hp NNs is their fast convergence to a stable minimum.

2.3.3 Constraint-Based Methods

JSSPs are conveniently formulated as either constraint satisfaction problems (CSPs) or constraint optimization problems (COPs). A CSP is defined by a set of variables and a set of constraints that restrict the values which can simultaneously be assigned to these variables. A COP is a CSP with an objective function to be optimized subject to the problem constraints. For JSSPs, the precedence constraints are that the operations in each job are performed in a particular order. The resource constraints are that each machine performs at most one task at any given time. Other constraints which may affect scheduling are release times, due dates, transfer times, setup times, resource availability,

etc. Some constraints are relaxable, such as due dates, frequency of tool changes, inventory levels, etc. To solve the CSP, a schedule is incrementally built by instantiating one scheduling decision (or variable) after another. One well-known example is OPIS [112], which uses domain-specific constraints, multiple layers of abstraction, and other scheduling heuristics to converge on near-optimal solutions. An important concept in constraint-based search methods is constraint propagation. Whenever a variable is instantiated, a new search state is created, where new constraints are used to deduce that some other unexplored values become inconsistent with the decision and must be pruned. Therefore constraint propagation reduces the amount of search needed to generate a schedule. However, constraint propagation cannot detect all inconsistencies during schedule construction. If a partial schedule is reached that cannot be completed without violating a problem constraint, one or several earlier assignments need to be altered. Such a process is called backtracking. Because the problem is NP-complete, backtrack search may require exponential time in the worst case. Research in CSP has produced several techniques to improve the efficiency of the backtrack search procedure [111]. Recently, intensive research on CSP and COP has applied constraint logic programming (CLP) to develop scheduling systems [25, 26, 27, 28, 29]. CLP is the result of generalizing logic programming unification to constraint solving over a computation domain. The declarative nature of CLP makes it easy to express the problem constraints. One advantage of CLP is its ability to capture incremental knowledge, reducing the search and handling the trade-offs between conflicting objectives.

2.

str

sui

wa

wit

alw

sta

sim

dep

whi

and

exe

spe

sch

feas

sche

exch

the e

for n

large

numb

2.4 Genetic Algorithm Methods

The use of GAs for JSSPs has been explored previously. "Classical" GAs use a binary string to represent a potential solution to a problem. Such a representation is not naturally suited for ordering problems such as the TSP and the JSSP, because no direct and efficient way has been found to map possible solutions 1:1 onto binary strings. Another problem with a "classical" GA representation is that simple crossover or mutation on strings nearly always produces infeasible solutions. Thus, previous researchers used some variations on standard genetic operators to address this problem.

Davis [31] was the first to suggest and demonstrate the feasibility of using GA's on a simple JSSP, *20x6 | R, s | ∑machine_cost*. A complete schedule is derived from a time-dependent preference list for each machine. A time is associated with each preference list which contains some permutation of contracts (groups of jobs), plus the elements "wait" and "idle". It is interpreted as showing which contract the machine should prefer to execute at the associated time or whether it should wait or stand idle. This problem-specific representation doesn't directly represent a schedule but instead is a decoder, or schedule builder, which performs the transition from chromosome representation to a feasible schedule. The schedule builder guarantees the feasibility and consistency of the schedules. Three domain-dependent operators are specified. The CROSSOVER operator exchanges preference lists for selected machines. The SCRAMBLE operator rearranges the elements of a selected preference list. The RUN-IDLE operator can insert idle times for machines. One disadvantage of this approach is that the representation is somewhat large. The total size of a solution is roughly the product of the number of machines, the number of time periods, and the number of jobs in the system.

Meanwhile, several GA-based approaches to TSPs found the introduction of domain knowledge to be the most important source of high performance operators and led to clues for more effective representations and operators for JSSPs [43, 44, 45]. Cleveland and Smith [32] investigated the applicability and effectiveness of a variety of the non-standard representations and operators introduced from GA work on TSPs. They focused specifically on a sector scheduling problem, which is a multi-stage, in-series problem, with a number of machines in parallel at each stage, and they made some simplifying assumptions to treat JSSPs as order sequencing problems. Unpredictable recirculation could occur in the job shop. Besides the ordered sequence representations, they also tested one direct representation, which specifies the starting time on the chromosome, and Davis's preference list. Some interesting results were found. First, the use of domain knowledge in GA operators was found to be variably effective at different stages of the search. Second, the GA was robust with respect to the unpredictable recirculation. Third, pure sequencing approaches were dramatically downgraded when the work-in-process times were included in the objective function. The direct representation and time-dependent preference list schemes, which contain the time information in the chromosome, got significantly better final solutions than the pure sequencing schemes.

Whitley *et al.* [33] reviewed the operators for TSPs and defined a new edge recombination operator. They applied this new operator to TSPs and obtained good results. However, they didn't get noticeably better results when applying the operator to more typical scheduling problems. Some of their conclusions are worth noticing. First, high performance operators should exploit the right information from the parent structure. Second, it's possible to achieve balanced optimization in a situation where GA's only

generate partial schedules, which are then evaluated based on the performance of the fully expanded schedules.

Syswerda [41] also implemented a similar method on a more realistic JSSP, which was represented as a TSP-like chromosome. He discussed the issues concerning the integration of domain knowledge into chromosome representation, interpretation, evaluation, and operators. In his approach, the schedule builder performs a local search and generates a feasible schedule with respect to the strong constraints. The knowledge of weak constraints and global evaluation factors are incorporated into the schedule evaluator. The global search is provided by a GA. He also mentioned that the direct schedule representation, although is cumbersome and needs complicated operators, has a definite advantage when dealing with complicated real-world problems. In the conclusion, he suggested a basic architecture for the GA-based approach, which provided for a mix of local and global search and a separation of the GA from the details of the problem. Another similar TSP-like approach can be found in [87].

Although JSSPs can be reduced to TSPs, some restrictions and simplifications need to be made and the TSP-like approaches have their limitations when applied to more complicated JSSPs. Some researchers have abandoned the TSP-like approach and developed more realistic representations and more complicated operators for JSSPs. Nakano and Yamada [35] used a binary representation to select the disjunctive arcs and conventional operators to generate new strings. Here the schedule builder not only produces schedules from feasible strings, but also applies a complicated repair procedure to find feasible schedules from infeasible strings. A treatment called *forcing* replaces an infeasible string with a similar, but feasible string. He applied this approach to three well-

known FT JSSPs, *6x6*| $|C_{max}$, *10x10*| $|C_{max}$, and *20x5*| $|C_{max}$ [8]. He found the optimal solution for the *6x6*| $|C_{max}$ problem, but didn't do well on the other two problems. He concluded that, first, a conventional GA can effectively solve JSSPs if the schedule builder is well designed. Second, the replacement of infeasible strings with feasible ones can help the population converge quickly, but too many replacements may cause premature convergence. The same idea of using a binary representation to select the disjunctive arcs is also found in Tamaki and Nishikawa's work [37] with a different repair procedure.

Bagchi *et al.* [34] and Uckun *et al.* [38] dealt with a more realistic JSSP, *JxM*| *R, a* $|\sum C_j$. A problem-specific indirect representation includes all information, *i.e.*, job order permutation, process plan for each job order, machines performing the operations specified in the plan. They applied problem-specific operators to ensure the validity of the chromosome. Thus, GAs do the all search and the schedule builder just creates a feasible schedule from the chromosome. Local hill climbing is used to improve the best chromosome discovered by the GA and the modified chromosome can replace the original chromosome. The results show that a representation comprising complete information ends up with a good solution, but the slower convergence due to the large search space prevents the applicability on real-world scheduling problems. Bruns [42] considered a similar problem, *JxM* | *R, r, a* | $\sum T_j^2$, but used a direct representation which includes all information as well as the start time. This approach also suffered from the problem of slow convergence.

Although the approaches discussed so far have developed many knowledge-augmented operators, none of these operators rely on any kind of theoretical foundation. Other approaches are based on the hybridization of GAs and some existing algorithms.

Ya
pr
to
at
ra
co
eff
alg
sar
sol
opt
fur
stat
solu

Ya
Pa
St
Do
Do
Ko
Kir

Yamada and Nakano [36] used operation completion times for their representation, and proposed a novel crossover operator, GA/GT, based on Giffler and Thompson's algorithm to ensures assembling valid and active schedules. The GA/GT crossover works as follows: at each decision point in the G&T algorithm (step 4 in Figure 2.6), one parent is selected randomly. However, in GA/GT crossover, the schedulable operation which has the earliest completion time reported in the parental schedule is chosen to be scheduled next. The effect of GA/GT crossover is the same as applying uniform crossover and using the G&T algorithm to interpret the resulting invalid chromosomes. They applied this scheme to the same three JSSPs as Nakano and Yamada [35] and achieved modest success. The optimal solutions of these three problems are sometimes found, but the chance of obtaining optimal solutions is still small. The authors suggested the use of local search heuristics for further improvements. Later, Davidor *et al.* [86] implemented this approach on a steady state parallel GA, a 2-dimensional grid structured population, and improved the average solution quality without additional computational cost.

Table 2.4 G&T-algorithm-based GA approaches

reference	representation	crossover
Yamada and Nakano [84]	completion time	uniform
Park and Park [85]	job order	modified uniform
Storer *et al.* [83]	perturbed processing time	standard
Dorndorf and Pesch [80]	starting time	standard
Dorndorf and Pesch [81]	priority rule	standard
Kobayashi *et al.* [82]	job order	subsequence exchange
Kim and Lee [89]	priority rule	PMX, OX, and PX

algo

The

appr

deco

appr

objec

for th

priori

used.

Pa

into G

geneti

or ?,

hasn't

is red

numb

each s

the re

metho

constra

promis

Far

Dorndorf and Pesch [80] encode the operation starting times and apply the G&T algorithm to decode the invalid offspring which are generated from standard crossover. The GA is integrated with a local search procedure. Other G&T-algorithm-based approaches are similar to the two above. The G&T algorithm is used as an interpreter to decode any offspring into an active schedule. Table 2.4 lists the G&T-algorithm-based GA approaches. The test problems are all static JSSPs with the minimizing makespan objective. These approaches are designed to transmit "useful characteristics" from parents for the creation of potentially better offspring. These "useful characteristics" can be priority rules or job sequences, depending on the representation and crossover methods used.

Paredis [39] also proposed a novel approach which embedded constraint propagation into GAs. The constraints are exploited in the generation of the initial population, and in genetic operators and evaluations. The representation is a ternary string, consisting of *0*, *1*, or *?*, which defines the selections of the disjunctive arcs. A *?* means that the selection hasn't yet been decided. By using the *?*, the probability of generating infeasible schedules is reduced. In this approach, each string represents a set of promising solutions. The number of promising solutions of one string depends on the number of *?*. The fitness of each string is derived from the objective values of the valid schedules in the set. Although the results were not good, this approach did show some ways to improve the GA-based methods. First, constraints can introduce background knowledge into GA's. Second, constraint propagation effectively reduces the search space and helps GA's explore promising regions.

Fang *et al.* [40] proposed a GA which uses a variant of an indirect representation

devis
unde
conv
perf
way
due
prob
the
from
trade
whic
the
fixes
appr
[59].
base
defin
Cros
GA
strate
other
appro

devised for the TSP. The schedule builder guarantees the validity of the schedule produced under crossover and mutation. In Fang's approach, some methods dealing with the gene convergence rate and the redundancy in the representation were applied to enhance the performance. They also considered the rescheduling problem and described two possible ways of rescheduling. One is rescheduling from scratch which is obviously to be avoided due to the large processing time required. Another is to construct a smaller scheduling problem from those affected parts of the current schedule. Such a method might preclude the discovery of an optimal schedule which might possibly be obtained by rescheduling from scratch. Clearly, a more sophisticated method should be developed to deal with the trade-off between schedule quality and processing time. Besides, their representation, which is indirect, exhibits one problem in the dynamic environment -- it's difficult to use the previous work done by the GA since the partial preservation of a schedule and local fixes of a schedule cannot be done on the chromosome level. Fang further tested the approach on a set of deterministic dynamic problems with different objective functions [59]. According to the results reported, the approach outperformed priority rules.

Another significant and recent result is that of Mattfeld *et al.* [90]. The approach is based on the earlier research of Bierwirth [91]. In this approach the representation is defined by a permutation with repetition, which only represents feasible schedules. Crossover is done by a generalization of OX. The approach was implemented on a parallel GA with social-like behavior of GA-individuals, which specified a local recombination strategy. The optimal solutions of the three FT JSSPs were found. They further tested other benchmark JSSPs and got impressive results. Bierwirth *et al.* [92] extend this approach to dynamic JSSPs. Stochastic dynamic JSSPs are decomposed into a sequence

of d

then

flow

rules

objec

2.5

P

real-ti

local

Althou

proble

determ

solutio

resourc

bottlen

produce

Som

(1)

of deterministic dynamic JSSPs. Whenever a job is released, a new problem is generated, then the GA solves the problem from a new initial population. The objective is the mean flow time. A simulation study shows that the approach clearly outperformed the priority rules. Rixen *et al.* [93] also applied this approach to dynamic JSSPs with a just-in-time objective.

2.5 Summary

Previous approaches either deal with simple, unrealistic JSSPs or take no account of real-time dynamic JSSPs. Priority rules and local search procedures only consider the local information and have strong limitations when the problems become complicated. Although constraint-based methods can effectively reduce the search space, several problems exist. First, the order in which the variables are instantiated is difficult to determine. Besides, different instantiation orders will limit search to other plausible solution space. For example, OPIS schedules high-contention resources before other resources, thereby limiting subsequence search to a subspace constrained by the bottleneck resource allocation procedure. Therefore, constraint-based methods tend to produce poor solutions in a large problem space.

Some problems in previous GA-based approaches are summarized as follows:

(1) Most GA-based research only studied the static JSSPs with the makespan objective. In dynamic JSSPs, which are more realistic, jobs can arrive at some known (deterministic JSSPs) or unknown (stochastic JSSPs) future times. Further, the importance of each job can be different and the objective is more complex. From a practical vantage, a large number of priority rule approaches have been pro-

posed and tested to address dynamic problems. While priority rules are computationally efficient and are useful for finding a reasonably good solution, prioritizing the jobs based on only a few job or machine parameters restricts the search capability. As was asserted by Adams *et al.* [94], given the computing power available today, it becomes more important to design effective approaches to obtain better schedules, even at additional computational cost.

(2) Most GA-based approaches need specific operators to ensure the validity of solution or to integrate the domain knowledge. Although the specific operators are difficult to design, if problem-specific knowledge is successfully incorporated into the operators, the GA can work more effectively on the particular problem. In JSSPs, the temporal relationships among all operations in a schedule are important. Simply working on the chromosome level usually focuses on only a small part of the schedule and overlooks the change of the temporal relationships in the whole schedule.

(3) Some problem-specific indirect representations suffer from the problem of false competition -- different representations of the same schedule competing against one another -- which is found in [89, 59]. In some indirect representations, such as prioritization of scheduling rules [81, 89], it is difficult to encode the schedule back to the chromosome. Furthermore, it's not easy to extend or adjust the indirect representation to a dynamic environment.

(4) Although Fang *et al.* [40] considered the rescheduling problem, a more elaborate method is necessary to deal with the trade-off between schedule quality and processing time. Besides, the dynamic JSSP studied is too simple and the approach

is not general to other nondeterministic events, such as new job releases, machine breakdowns, or change of job priority.

(5) The rescheduling approach in [92] is done by discarding the old population and then constructing the new schedule from scratch. Consider the final population in the last time period before new jobs arrive. Because typically only a few operations are removed from the last (deterministic) job shop problem to generate the new problem, only a small fraction of the information in the population has changed. The old population still contains useful information. Thus it is reasonable to create an initial population by modifying the already adapted individuals to the needs of the new problem, and then to allow the GA to continue the search based on the modified population. No previous researcher has implement this idea.

(6) The manufacturing environment studied is too simple. No previous researchers investigated the influence of the manufacturing environment, such as different machine workloads, imbalance of machine workload, and different flow allowances.

As

mainta

and ge

schedu

algorit

validity

with de

JSSP is

GA. F

popula

3.1 S

Clas

represe

because

binary st

on string

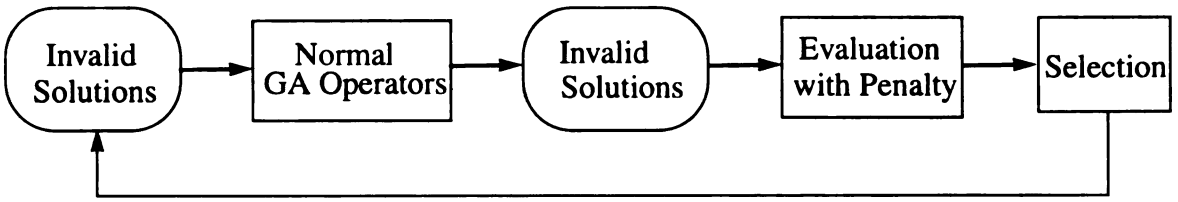
CHAPTER 3

The GA-based Scheduling System

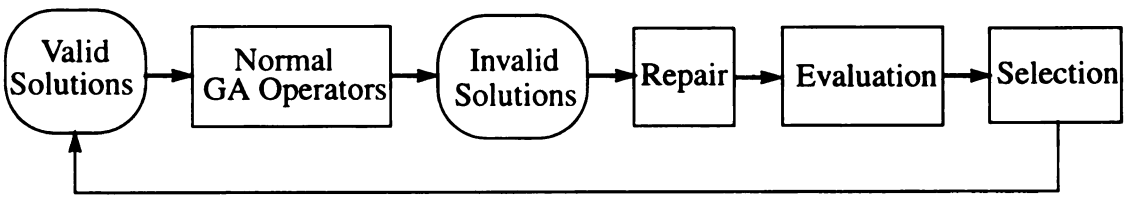
As discussed in the previous chapter, when solving JSSPs using GAs, it is difficult to maintain the validity of the solutions. This problem arises in the design of representation and genetic operators, and affects other components of GAs. In this chapter, the basic scheduling system for static JSSPs is described in Section 3.1. Based on the G&T algorithm, two new operators, THX crossover and mutation, are proposed. The problem of validity is also discussed. In Section 3.2, the basic scheduling system is extended to deal with deterministic dynamic JSSPs. For stochastic dynamic JSSPs, a deterministic dynamic JSSP is generated at each occurrence of a nondeterministic event, and then solved by the GA. Furthermore, an innovative rescheduling method, which modifies the adapted population into a new population between successive events, is proposed.

3.1 Static Model

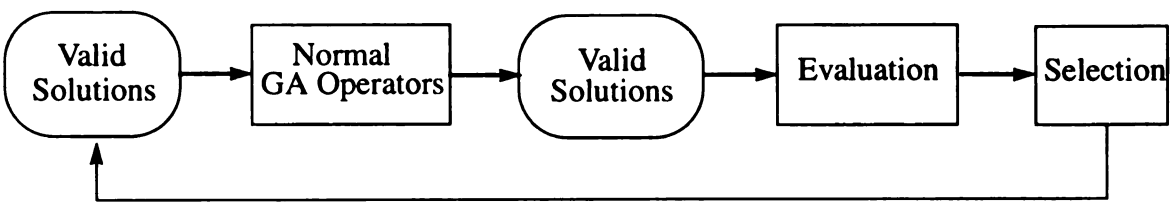
Classical GAs use a binary string to represent a potential solution to a problem. Such a representation is not naturally suited for ordering problems such as TSPs and JSSPs, because no direct and efficient way has been found to map possible solutions 1:1 onto binary strings. Another problem with classical GAs is that simple crossover or mutation on string-representing schemes nearly always produces infeasible solutions. Four types of



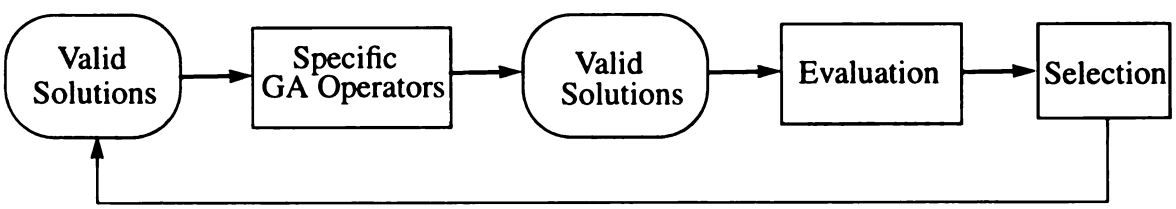
(a)



(b)



(c)



(d)

Figure 3.1 Four types of approach for the validity problem

approach to this problem are shown in Figure 3.1.

Type (a) uses normal operators and allows invalid solutions in the population. In the evaluation stage, the invalid solutions are penalized with a relatively bad fitness so the invalid solutions do not tend to survive in the selection stage. Two problems exist in this approach. First, it is not easy to construct the penalty function. The penalty function should be harsh enough so that the GA does not converge to invalid solutions. But, if the penalty function is too severe, the information provided by invalid solutions will be lost. Care must be taken to find a balance between the validity of solutions and preservation of information. Another problem of this approach is that if the space of valid solutions is very small compared to the whole representation space, the probability of finding one valid solution is very small. Thus, the GA will waste most of its time on invalid solutions. This type of approach is inefficient for JSSPs because of the second problem.

Type (b) also uses normal operators and allows invalid solutions in the population, but it uses a "repair" operator to transform the invalid solutions into a similar valid solutions. In general this is done by a small modification of the representation. Some examples are Nakano and Yamada [35], Yamada and Nakano [84], and Dorndorf and Pesch[80].

Type (c) uses normal operators and only allows valid solutions in the population. The validity problem is addressed in the representation scheme so the chromosomes always represent valid solutions. Some examples are Dorndorf and Pesch [81] and Storer *et al.* [83].

All the previous types of approach use normal operators. Most of the approaches to JSSPs to date are type (d), which develop specific operators and only allow valid solutions in the population. The design of problem-specific operators focuses on the integration of

domain knowledge and the transmission of "useful characteristics" from parents to offspring. The scheduling system for static JSSPs developed in this dissertation is of type (d) and is described in the following subsections.

3.1.1 Representation

Two approaches have been used to deal with the problem of representation. The first is an *indirect* representation, which encodes a string of instructions to a schedule *builder.* Some examples of an indirect representation are job order permutation and prioritization of scheduling rules. In these schemes, the schedule builder guarantees the validity of the schedules produced. Another approach is to use a *direct* representation which encodes the *schedule itself.* Some examples of direct representations are the encoding of the operation completion times or the operation starting times.

Our approach uses a direct representation, which encodes the operation starting times. The number of the fields on the chromosome is the number of operations. Figure 3.2 shows a general representation. Figure 3.3 gives an example of the representation of the schedule in Figure 2.2. Such a direct representation doesn't suffer from the problem of false competition (Section 2.5). Another advantage in the direct representation is the ease of encoding the schedule into the chromosome. In some indirect representations, such as prioritization of scheduling rules, it is difficult to encode the schedule back to the

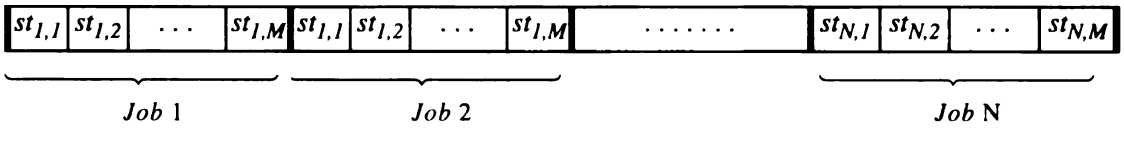
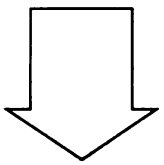
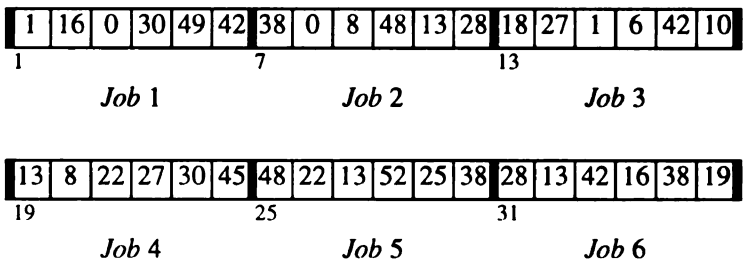


Figure 3.2 General representation of JSSPs

Starting Time / Job	Machine 1	2	3	4	5	6
1	1	16	0	30	49	42
2	38	0	8	48	13	28
3	18	27	1	6	42	10
4	13	8	22	27	30	45
5	48	22	13	52	25	38
6	28	13	42	16	38	19

Starting time of each operation in the example of Figure 2.2





Chromosome representation

Figure 3.3 Example of representation

chr

ger

enc

3.1

inv

app

inv

des

wh

JS

Sir

sch

cor

the

cro

of

cro

inf

are

cros

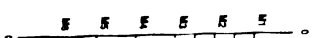
chromosome. This advantage is more important in our approach because the designed genetic operators work on the schedule level and the modified schedules need to be encoded back to the chromosomes.

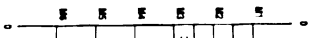
3.1.2 Crossover

In the direct representation, not every encoding corresponds to a valid schedule. If invalid encodings are allowed in the population, a type (a), (b), or (c) approach can be applied to maintain the validity of the schedule. Because our approach doesn't allow invalid encodings in the population, we must design specific genetic operators as described in the type (d) approach.

The crossover operator is inspired by the G&T algorithm. Some related approaches which are G&T-algorithm-based have been briefly reviewed in the previous chapter. In JSSPs, the temporal relationships among all operations in a schedule are important. Simply working on the chromosome level usually focuses on only a small part of the schedule and overlooks the change of the temporal relationships in the whole schedule. In contrast to previous approaches, which work on the chromosome level, we have designed the time horizon exchange (THX) crossover, which works on the *schedule* level. THX crossover randomly selects a crossover point just as a standard crossover does, but instead of using the crossover point to exchange two chromosomes, THX crossover uses the crossover point as a scheduling decision point in a G&T algorithm to exchange information between two schedules.

Figure 3.4 shows an example of THX crossover in the FT 6x6 problem. The schedules are presented in a machine Gantt chart. The portion of the child schedule before the crossover point is exactly the same as in one parent. The temporal relationships among





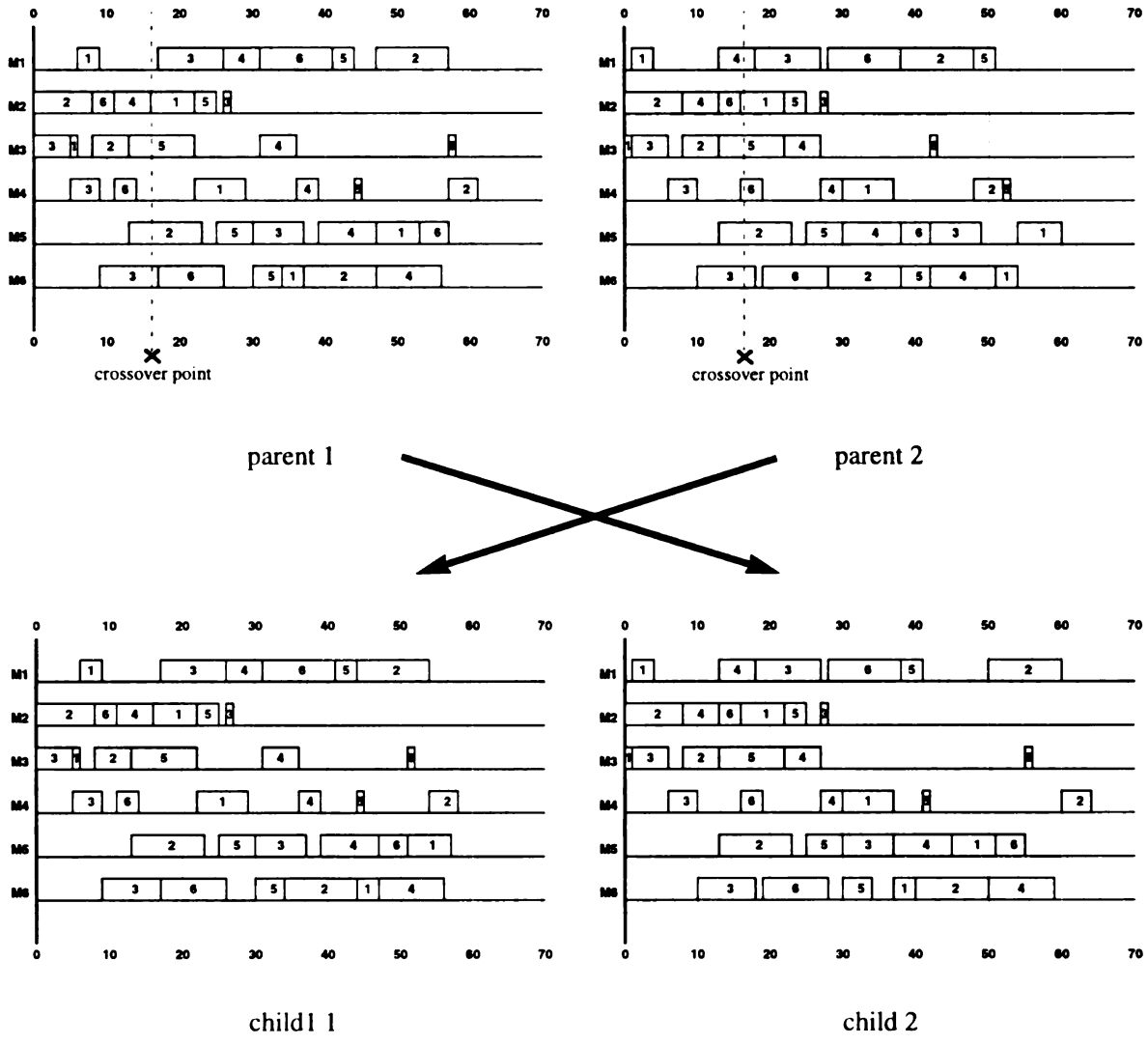


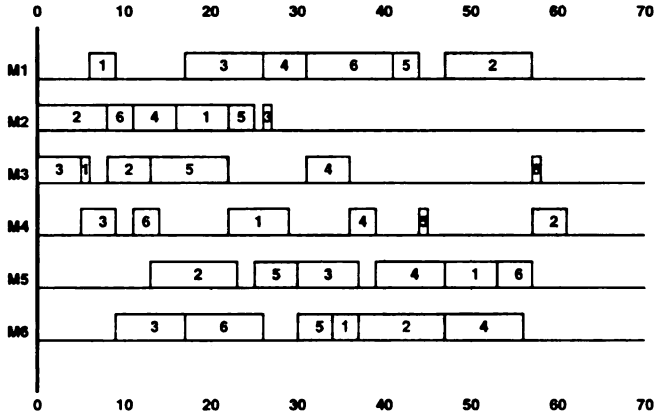
Figure 3.4 An example of THX crossover

operations in the remaining portion are inherited from the other parent to the extent possible (*i.e.*, while maintaining a valid schedule).

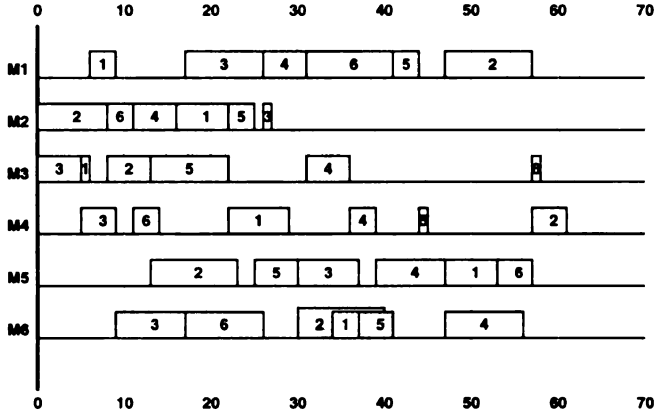
3.1.3 Mutation

Another important operator in GAs is mutation. The mutation operator is based on the disjunctive graph of the schedule. Although exchanging a single pair of adjacent tasks which are on the same machine and belong to a critical path can preserve the acyclic property of the directed graph, the number of child schedules that are better than the parent tends to be very limited, as was observed by Grabowski *et al.* [95]. They defined a *block* as a sequence of successive operations on the critical path which are on the same machine with at least two operations. The reversal of a critical arc can only yield an immediate improvement if at least one of the reversed operations is either the first or the last operation of the block (however, other exchanges may be needed before the step which actually yields an improvement). Thus our mutation focuses on the block. Two operations in the block are randomly selected and reversed. After two operations are reversed, the resulting child might be valid or invalid. We then apply the G&T algorithm to interpret the child. Unlike type (b) approach for the validity problem, which only repairs invalid solutions to valid solutions, the valid child is also interpreted by the G&T algorithm and transformed into an active schedule. In this mutation scheme, no cycle detection is needed. Furthermore, the G&T algorithm guarantees that the two selected operations are reversed in the new schedule and that the new schedule is active.

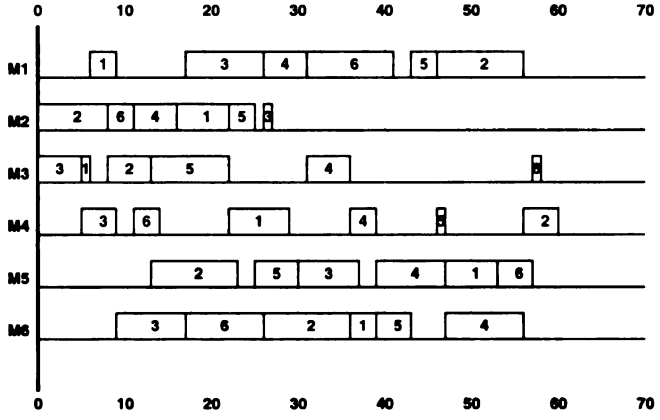
Figure 3.5 shows an example of the mutation. Figure 3.5 (a) is the parent before mutation. The operations belonging to the critical path are shaded in light gray. There are two blocks in the schedule. One is operation (2, 3) and (5, 3). The other is operation (5, 6),



(a) Before mutation



(b) Swap two operations in one block



(c) After interpretation of the schedule in (b) by the G&T algorithm

Figure 3.5 An example of mutation

(1, 6).

resulti

two op

Figure

that on

other c

after th

3.1.4

In

minima

most s

selecte

JSSPs.

3.2

Th

after a

3.2.1

Th

Assum

algorit

(1, 6), and (2, 6). Assume operation (5, 6) and (2, 6) are selected and swapped. The resulting schedule is given in Figure 3.5 (b). Note that the schedule is invalid. After the two operations are swapped, the schedule is then interpreted by the G&T algorithm. Figure 3.5 (c) shows the child schedule, which is not only valid but also active. Note also that only operations (5, 6) and (2, 6) are reversed and the temporal relationships among other operations are preserved. In this example, an improvement of makespan is made after the mutation.

3.1.4 Objective Functions

In static JSSPs, makespan is a good performance measure. The schedule with the minimal makespan often implies a high utilization of machines. Because the objective of most static JSSP benchmarks is to minimize the makespan, the makespan objective is selected for comparison and to assess the performance of the scheduling system in static JSSPs.

3.2 Dynamic Model

The scheduling system described in the Section 3.1 can be applied to dynamic JSSPs after a few modifications, which are presented in the following subsections.

3.2.1 Genetic Operators

The two genetic operators, THX crossover and mutation, are G&T-algorithm-based. Assume the release time of job j is r_j. For deterministic dynamic JSSPs, step 1 of the G&T algorithm is modified to take account of the job release times as follows:

```
Step 1:
     Let C contain the first schedulable operation
```

Af

dynam

whene

consid

Assum

into ac

3.2.2

Th

into a

the jo

job j f

being

the re

arriva

exam

Mach

which

remo

of each job;
Let $r_{jm} = r_j$, for all operations (j, m) in $\boldsymbol{C}$.

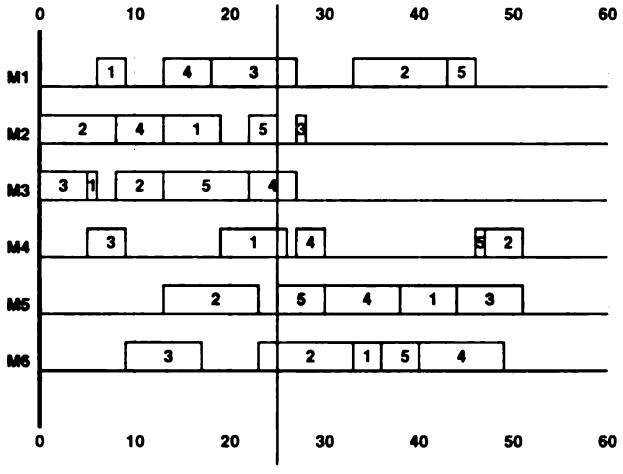
After the modification, the genetic operators are able to deal with deterministic dynamic JSSPs. For stochastic dynamic JSSPs, a deterministic problem is generated whenever a new job enters the system. Besides the job release time, it is also necessary to consider the machine blocked-out time, which is discussed in the next subsection. Assuming the machine is available at time a_m, we modify step 1 of G&T algorithm to take into account the machine blocked-out times as follows:

Step 1:
Let $\boldsymbol{C}$ contain the first schedulable operation of each job;
Let $r_{jm} = \max(r_j, a_m)$, for all operations (j, m) in $\boldsymbol{C}$.

3.2.2 Time Decomposition Method

The method, proposed by Raman *et al.* [56], decomposes the stochastic dynamic JSSP into a sequence of deterministic dynamic JSSPs. Whenever a new job enters the system, the job information is updated. If job j is completed before that point in time, we remove job j from the system. If only some operations of job j are completed before that point or being processed at that point, we modify job j by removing these operations and update the release time of job j. Because one or more machines can be busy at the time of a job arrival, such machines are blocked out for the period of commitment. Figure 3.6 shows an example of the time decomposition method. A new job 6 arrives at the system at time 25. Machines 1, 3, 4, and 6 are blocked out because they are busy at time 25. Any period in which a machine is blocked-out is shaded in gray in Figure 3.6 (b). Some operations are removed from the system and the information for job 6 is added to the new problem. The

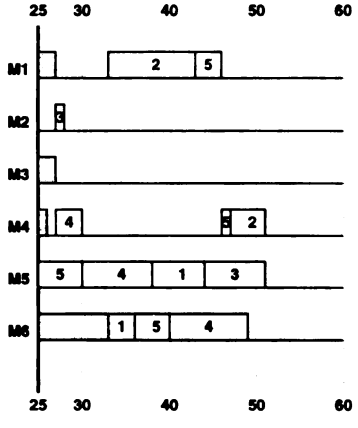

0
M1
M2
M3
M4
M5
M6
0



Job	Release time	Machine routing (processing time)
1	0	3(1) 1(3) 2(6) 4(7) 6(3) 5(6)
2	0	2(8) 3(5) 5(10) 6(10) 1(10) 4(4)
3	0	3(5) 4(4) 6(8) 1(9) 2(1) 5(7)
4	0	2(5) 1(5) 3(5) 4(3) 5(8) 6(9)
5	0	3(9) 2(3) 5(5) 6(4) 1(3) 4(1)

Machine	1	2	3	4	5	6
Available time	0	0	0	0	0	0

(a) The deterministic problem before job 6 arrives



Job	Release time	Machine routing (processing time)
1	26	6(3) 5(6)
2	33	1(10) 4(4)
3	27	2(1) 5(7)
4	27	4(3) 5(8) 6(9)
5	25	5(5) 6(4) 1(3) 4(1)
6	25	2(3) 4(3) 6(9) 1(10) 5(4) 3(1)

Machine	1	2	3	4	5	6
Available time	27	25	27	26	25	33

(b) The deterministic problem after job 6 arrives

Figure 3.6 An example of time decomposition method

job rel

3.2.3

In s

arrival.

resche

individ

system

which

period

from th

fractio

an initi

proble

popula

an inn

G&T

proble

proble

The o

modifi

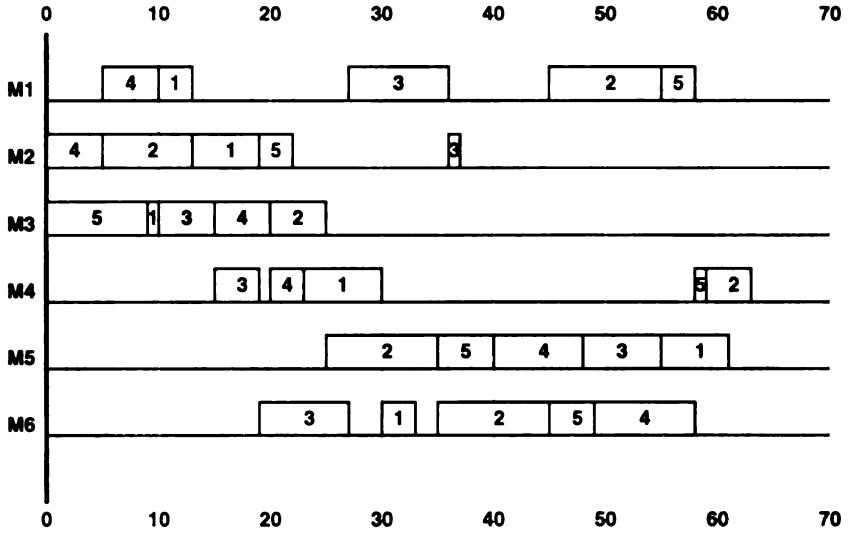
Step

job release times and machine available times are updated.

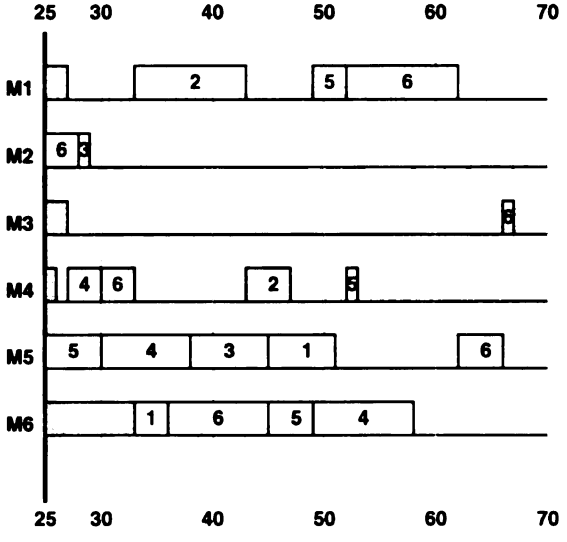
3.2.3 Rescheduling Method

In stochastic JSSPs, after a new deterministic problem is generated at the time of a job arrival, the current schedule needs modifications. Two methods can address the rescheduling problem. One is to discard the old population and construct the new individuals (schedules) from scratch. This can be done simply by restarting the scheduling system with the new deterministic JSSP. The other method uses a special feature of GAs which was observed by Bierwirth *et al.* [96]. Consider the final population in the last time period before the new jobs arrive. Because typically only a few operations are removed from the last (deterministic) job shop problem to generate the new problem, only a small fraction of the information in the population has changed. Thus it is reasonable to create an initial population by modifying the already adapted individuals to the needs of the new problem and then to allow the GA to continue the search based on the modified population. Bierwirth *et al.* did not do further investigation on this idea. Here we propose an innovative method to modify the adapted population. This method is also based on the G&T algorithm. When constructing an individual in the initial population of the new problem, the temporal relationships among the operations which are also in the last problem are inherited from an individual in the adapted population to the extent possible. The operations of the new job(s) are randomly scheduled among the old jobs. This modification process is implemented by changing step 4 of the G&T algorithm as follows:

Step 4:
Randomly select one operation from ***G***.
If the operation is from the new job(s), schedule



(a) A schedule in the last population before job 6 arrives



(b) The modified schedule of (a) in the initial population of the new problem

Figure 3.7 An example of modifying a schedule from the last population to the initial population of the new problem

```
        it;
    else schedule the operation in G which is from the
         old problem with the earliest starting time
         reported in the individual of the adapted
         population.
```

Figure 3.7 shows an example of modifying an individual from the last population to the initial population of the new problem. This example follows the example in Figure 3.6. The operations of the new job 6 are inserted among the other operations as shown in Figure 3.7 (b). Notice that the temporal relationships among the old operations in Figure 3.7 (a) are preserved to the extent possible.

The two rescheduling methods can work together to achieve the superior results. If there is enough time, rescheduling primarily from scratch, but with a small proportion of the population consisting of modified individuals, can do a more thorough search without losing all information found in the old population. If the old population is mostly converged, rescheduling primarily from modified individuals with some proportion of the population generated from scratch can increase the diversity in the new population. The proportions of the population generated from these two methods should be problem dependent.

3.2.4 Objective Functions

As described in subsection 2.2.2, reducing turnaround time through the shop or reducing the amount of tardiness is usually the primary objective in dynamic JSSPs. Therefore, we consider the objective functions shown in Table 3.1. For reporting purposes, we use the normalized value of the objective, which is also defined in Table 3.1. Except for the objective of weighted flow time, these objective functions are due-date-related. Also notice that the weighted earliness plus weighted tardiness is a nonregular objective

function.

Table 3.1 Objective functions in dynamic JSSPs

Objective Function	Definition	Definition (Normalized)
Weighted Flow Time	$\sum_j w_j(C_j - r_j)$	$\left(\sum_j w_j(C_j - r_j)\right) / \left(\sum_j w_j p_j\right)$
Maximum Tardiness	$\max_j T_j$	$\left(\max_j T_j\right) / \left(\sum_j p_j\right)$
Weighted Tardiness	$\sum_j w_j T_j$	$\left(\sum_j w_j T_j\right) / \left(\sum_j w_j p_j\right)$
Weighted Lateness	$\sum_j w_j L_j$	$\left(\sum_j w_j L_j\right) / \left(\sum_j w_j p_j\right)$
Weighted Number of Tardy Jobs	$\sum_j w_j U_j$	$\left(\sum_j w_j U_j\right) / \left(\sum_j w_j\right)$
Weighted Earliness plus Weighted Tardiness	$\sum_j w_j(E_j + T_j)$	$\left(\sum_j w_j(E_j + T_j)\right) / \left(\sum_j w_j p_j\right)$

CHAPTER 4

Parallel GA Architectures

One common problem in classical GAs is *premature convergence* -- *i.e.*, inability to search beyond local minima. While classical GAs are more resistant to premature convergence to local minima than other local search procedures (Section 2.3.2), GAs are not immune. One approach to reduce the premature convergence of a GA is parallelization of the GA into disjoint subpopulations, which is also a more realistic model of nature than a single population. Parallel GAs (PGAs) may be implemented using parallel processors or processes, or may be done in a single process, via time-shared execution of the code for each subpopulation. The advantages of a PGA are thus - 1) fighting premature convergence, and 2) allowing speedup via use of parallel processors. These processors need only be loosely coupled, which is another advantage of PGAs as an optimization method.

Currently, there are two kinds of PGAs that are widely used: *coarse-grain* GAs (cgGAs) and *fine-grain* GAs (fgGAs). This chapter described the premature convergence problem and these two PGAs. Two new hybrid models are also presented in this chapter.

4.1 Premature Convergence Problem

To understand premature convergence, we first examine the concept of *genetic drift*.

Genetic drift is the modification of a population resulting from random events. Under no selection pressure (a random walk), a population will be dominated by genetic drift and converge [97]. So whether selection or drift dominates, convergence is inherent in classical GAs and is the reason that classical GAs can not maintain different high fitness individuals in one population. Once a suboptimal individual dominates the population, selection is likely to keep it there and prevent further adaptation within any practical timeframe. This is the premature convergence problem.

Previous research has focused on two general approaches to avoid premature convergence. The first approach is to lower the convergence speed so the GA can do a more thorough search before converging, increasing the chance of finding the global optimum. This scheme often makes modifications in the fitness scaling stage or selection stage as described in Section 2.1. The second approach focuses on keeping the diversity of the population high by modifying the traditional replacement scheme and genetic operators, as described in the following examples.

De Jong introduced the concept of a *crowding scheme* [98]. In a crowding scheme, an offspring replaces an existing individual according to its similarity with other individuals in a randomly drawn subpopulation of size CF (crowding factor). The similarity measure is based on the Hamming distance between solutions in the subpopulation. Similar individuals compete in the same subpopulation, reducing the speed at which one individual can dominate the whole population.

Mauldin introduced a *uniqueness operator* to maintain high genotype diversity [99]. The insertion of an offspring into the population is possible only if the offspring is genotypically different from all other individuals of the population (specified as a given

Hamming distance).

Goodman introduced a form of mating restriction, called *incest reduction*, to promote diversity, while preserving all members of the selection-biased pool (or their offspring) into the next generation [114]. In this scheme, after the individuals are selected to produce offspring for the next generation, pairs for crossover are picked by randomly choosing the first parent from the survivors, without replacement, then randomly choosing a number of candidates for the second parent from the survivors. The Hamming distance of each candidate from the first parent is calculated, and the one with the greatest Hamming distance is picked for the crossover. The two selected parents are removed from the list of survivors for subsequent crossovers after the crossover is completed.

Goldberg defined a *sharing function* [100] that permits the formation of stable subpopulations centered on different individuals, thereby permitting the parallel investigation of many peaks in the search space. This scheme determines the selection probability according to the average fitness of similar individuals in the population. The similarity criterion can be the distance between either encoded genotypes or decoded phenotypes.

Crow used a *phenotypic comparison function* to enforce restricted mating [101]. Only similar individuals are allowed to mate. Although this scheme is used to fight the "lethals" problem -- *i.e.*, the production of low fitness offspring which could be produced from the mating of incompatible individuals, it also works against diversity.

Although all of the above schemes partially alleviate the convergence problem, some are not appropriate for real-world problems. First, a similarity evaluation between all individuals is computationally expensive and therefore degrades performance.

Furthermore, these evaluations are sequential in nature and hard to parallelize, a disadvantage for some GA implementations. Second, a phenotypic similarity evaluation is domain specific, requiring a special encoding for each new problem. These encodings are necessarily domain-dependent and not easily applied to other problems. Among the previous schemes, some are not computationally expensive, such as DeJong-style crowding and incest reduction. These schemes can be easily applied to single-population GAs without degrading the performance. It is clear that some PGA models such as cgGAs (described below) can incorporate these schemes in each node to further maintain the diversity and improve the performance.

An alternative approach to deal with premature convergence is parallelization of GAs. The PGA produces a more realistic model of nature than a single-population GA. Unlike some examples in the previous two approaches, which pay a high computational cost for maintaining subpopulations based on similarity comparisons, PGAs retard premature convergence by maintaining multiple, somewhat separate subpopulations which may be allowed to evolve more independently (or, more precisely, by employing non-panmictic mating). This allows each subpopulation to explore different parts of the search space, each maintaining its own high-fitness individuals, and allows control of how mixing occurs among subpopulations, if at all.

4.2 Coarse-grain GAs

cgGAs, also called island-parallel GAs, maintain distinct subpopulations, each of which acts as a single-population GA [47, 102, 103, 104]. In general, each subpopulation contains a large number of individuals. Subpopulations may or may not be divided among

more than one node (a process or processor). All subpopulations may, of course, be run within a single process by sequencing through them in some fashion or by time sharing their workloads. At certain intervals, some individuals can migrate from one subpopulation to another. While not an exhaustive description, cgGAs are categorized along three dimensions: migration method, connection scheme, and node homogeneity.

4.2.1 Migration Method

The migration method determines how often, and under what timing constraints, individuals are exchanged between subpopulations.

(1) Isolated Island GAs

There is no migration between subpopulations. This is the simplest model of cgGAs.

(2) Synchronous Island GAs

The migration between two subpopulations is synchronized. By synchronizing migration, the subpopulations evolve at roughly the same "rate" before exchanges occur. This rate measure (number of generations, convergence percentage, etc.) determines the synchronicity of the subpopulations. Dedicated parallel hardware can directly support such synchronization, but synchronization in a distributed workstation environment can cause uneven work loads. Different machine speeds and different loads can cause some nodes to "wait", slowing the speed of evolution to that of the slowest node.

(3) Asynchronous Island GAs

Asynchronous GAs allow migration based on a single event that does *not* relate to the state of evolution across all of the system's subpopulations. Such asynchronous behavior is the kind of migration typically found in nature, since different environments are responsible for differences in evolution speed. Our recent work has focused on

asynchronous GAs, since we implement cgGAs on distributed workstations in a shared-use (campus computing) environment. In this setting, each GA process competes for computing resources with whatever users/processes are on the same machine. The different loads and diverse machine architectures cause different evolution speeds in each node, making asynchronous migration more appropriate. This is especially true when the number of processing nodes is large, and the machine architecture types diverse, as occurs in MSU's cgGA runs. From our previous research [47], the insertion of a relatively high-fitness individual from a fast-evolution node into a low-fitness population on a slow-evolution node helps the low-fitness subpopulation find the global optimum of a graph partition problem, for example. Although further investigation is needed, the preliminary results show asynchronous PGAs to be a promising approach.

4.2.2 Connection Scheme

The connectivity of the processing nodes, in terms of the degree of connectivity and the topology of connection, affects the performance of a cgGA. More interestingly, another question is how such connections might be allowed to change over time.

(1) Static Connection Scheme

The connections between nodes are established at the beginning of the run and are not modified, keeping the network topology static during execution. Topologies can be of various types, including: lines, rings, n-cubes, etc., but the topology determines which neighbors can exchange information, and that topology is static.

(2) Dynamic Connection Scheme

The topology of node connection is mutable during runtime. There are two basic reasons to allow modification of the topology during runtime. First, such modifications

make distributed workstation parallelism practical in a real-world environment. If a node's GA process is stopped, then reconfiguration of the network occurs so as to continue processing. cgGAs can withstand a number of such events before losing evolutionary progress. Second, reconfiguration could occur based on changes that occur in the evolution of the subpopulations. One of the drawbacks of cgGAs is that the insertion of a new individual from another subpopulation may not be effective. The new individual may be grossly incompatible with that subpopulation, and therefore either be ignored or dominate the subpopulation. To avoid this, the subpopulations could start processing without any neighbors, and when a migration occurs, the node's neighbors could be determined by similarity (or dissimilarity) with other subpopulations (whichever may be appropriate for the problem at hand). For example, the best individuals of each population can be compared, and those with the closest Hamming distance could be established as neighbors. This establishes a *distance connection* topology that changes over time, and maintains interchange between only similar (or dissimilar) subpopulations.

4.2.3 Node Homogeneity

Node homogeneity is a measure of how similar the GA processes are on different processing nodes.

(1) Homogeneous Island GAs

The GA on each node uses the same parameters (population size, crossover rate, mutation rate, migration interval, etc.), genetic operators, objective functions, and encoding methods. Most research on PGAs has focused on homogeneous PGAs. One advantage of homogeneous island GAs is that they are relatively easily implemented.

(2) Heterogeneous Island GAs

dit

ge

tru

set

imp

par

enc

can

4.2.

V

iiGA

run

migr

4.2.

C

solu

be al

bit ir

such

repre

repres

Heterogeneous Island GAs allow the evolution of subpopulations via GAs with different parameters, genetic operators, objective functions, and/or encoding methods. In general, it is difficult to find the best initial parameter settings for a GA. This is especially true in the case of multiple objective optimization problems where various parameter settings can cause a GA to focus search in different portions of the search space. More importantly, many problems can be encoded in a GA using different methods, each with particular advantages and disadvantages. Such variations in parameter settings and encoding methods pose problems for interchange between populations, but these problems can be addressed, and have been addressed successfully, for example, in [47, 113].

4.2.4 Injection Island GAs (iiGAs)

We have developed a new PGA architecture called injection island GAs (iiGAs). iiGAs are a class of asynchronous, static- *or* dynamic-topology, heterogeneous GAs. We run them on a distributed network. The two most interesting aspects of an iiGA are its migration rules and the heterogenous nature of its nodes.

4.2.4.1 iiGA Heterogeneity

GA problems are typically encoded as an n-bit string which represents a complete solution to the problem. However, for many problems, the resolution of that bit string can be allowed to vary. That is, we can represent those n bits in n' bits, $n' < n$, by allowing one bit in the n'-long representation to represent r bits, $r>1$, of the n-long bit representation. In such a translation, all r bits take the same value as the one bit from the n'-long representation. Thus the n'-long representation is an abstraction of the n-long representation. More formally, let

$n = p \times q$ where p and q are integers, $p, q \geq 1$.

Once p and q are determined, we can re-encode a *block* of bits p' x q' as 1 bit if and only if

$p = l \times p,\ q = m \times q$ where l and m are integers, $l, m \geq 1$.

Such an encoding has the following basic properties,

(1) The smallest block size is $1x1$. The search space is 2^n.

(2) The largest block size is pxq. The search space is $2^1=2$.

(3) The search space with a block size $p'xq'$ is $2^{p/p'} \text{x } 2^{q/q'}$.

In general, an iiGA has multiple subpopulations that encode the same problem using different representations which are not necessarily defined in the block structure. All that is required is the mapping function which transforms any subpopulation's representation into the representation in a more detailed level. This mapping function then allows migrations from a low resolution to a high resolution node as described in Section 4.2.4.2.

4.2.4.2 iiGA Migration Rules

An iiGA may have a number of different block sizes being used in its subpopulations. To allow interchange of individuals, we only allow a one-way exchange of information, where the direction is from a low resolution to a high resolution node. Solution exchange from one node type to another requires translation to the appropriate block size, which is done without loss of information from low to high resolution. One bit in an n'-long representation is translated into r bits with the same value in an n-long representation. Thus all nodes *inject* their best individual into a higher resolution node for "fine-grained" modification. This allows search to occur in multiple encodings, each focusing on

different areas of the search space.

More formally, we note that node x with block size $p_1 x q_1$ can pass individuals to node y with block size $p_2 x q_2$ if and only if

$$p_1 = j \times p_2 , \quad q_1 = k \times q$$

where j, k are integers, $j, k \geq 1$.

This establishes a hierarchy of exchange, where node x (lower resolution) is the parent of node y (higher resolution) and node y is the child of node x. The direct child is the child with the largest block size; others are "heirs" of the lower-resolution "ancestors." Based on this general migration rule, we have designed the four following static topologies:

- Simple iiGAs

  Each node passes individuals to only one node, the node with the highest resolution (a block size of 1x1).

- Complete iiGAs

  Each node passes individuals to all of its heirs (of higher resolution) in the hierarchy.

- Strict iiGAs

  Each node passes individuals only to its direct children.

- Loose iiGAs

  Each node passes individuals to both its direct children and the node with the highest resolution (block size of 1x1).

4.2.4.3 iiGA Advantages

iiGAs have the following advantages over other PGAs.

(1) Building blocks of lower resolution can be directly found by search at that resolution. After receiving lower resolution solutions from its parent node(s), a node of higher resolution can "fine-tune" these solutions.

(2) The search space in nodes with lower resolution is proportionally smaller. This results in finding "fit" solutions more quickly, which are injected into higher resolution nodes for refinement.

(3) Nodes connected in the hierarchy (nodes with a parent-child relationship) share portions of the same search space, since the search space of parent is contained in the search space of child. Fast search at low resolution by the parent can potentially help the child find fitter individuals.

(4) iiGAs embody a divide-and-conquer and partitioning strategy which has been successfully applied to many problems. Homogeneous PGAs cannot guarantee such a division since crossover and mutation may produce individuals that belong to many subspaces -- i.e., the divisions cannot be maintained. In iiGAs, the search space is fundamentally divided into hierarchical levels with well defined overlap (the search space of the parent is contained in the search space of the child). A node with block size $r = p' \times q'$ only searches for individuals separated by Hamming distance r.

(5) In iiGAs, nodes with smaller block size can find the solutions with higher resolution. Although DPE [123] and ARGOT [124] also deal with the resolution problem using zoom or inverse zoom operators, they are different from iiGAs. First, they are working on the phenotype level and only for real-valued parameters. iiGAs divide the string into small blocks regardless of the meaning of each bit.

4.

fg

ma

env

is n

Mac

popu

para

each

requ

foun

"neig

stron

indiv

deter

popul

indivi

Second, the sampling error can fool them into prematurely converging on sub-optimal regions. Unlike PDE and ARGOT, iiGAs search different resolution levels in parallel and reduce the risk of searching the wrong target interval.

4.3 Fine-grain GAs

Instead of running a number of GAs in parallel, fgGAs parallelize the GA itself. fgGAs are motivated by nature. In nature, there is no global selection and no panmictic mating. Natural selection and natural mating take place in an individual's local environment. Because each individual evolves only based on local information, this model is naturally parallel and it is easy to make use of parallel machines such as the Connection Machine [125], DAP [126], and Transputer [127].

In fgGAs, individuals are spatially arrayed in some manner and an individual in the population can interact only with individuals "close" to it [105, 106]. If implemented on parallel hardware, then in contrast to cgGAs, only one or a few individuals are assigned to each processor. Migration between neighbors or mating outside the local subpopulation is required, and the frequency of migration between neighbors is typically much higher than found in cgGAs. The topology of individuals in the "template" which defines the breeding "neighborhood" determines the degree of isolation from other individuals and therefore strongly influences the diversity of the individuals in the population. Therefore, each individual is, in effect, part of multiple subpopulations, with membership typically determined by the network connection topology of the processors. Thus the entire population can be viewed as numerous, small, overlapping subpopulations, and all the individuals can be considered to be continuously moving around within their

neigh

neigh

susce

will p

can sl

4.4

Tw

embe

subpo

of mi

within

each t

occas

Th

the co

relativ

node

neighb

in wh

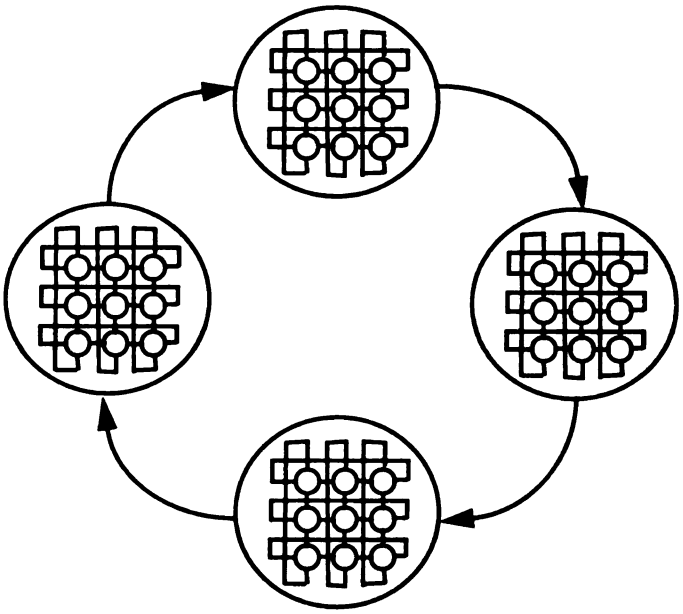
neighborhoods, so that global communication is possible, but not instantaneous.

The main concern in fgGAs is the network topology. High connectivity between neighbors increases the spread of high-fitness individuals, making subpopulations susceptible to domination and perhaps premature convergence. Limiting the interactions will partially solve this problem, but essentially reduces the size of subpopulations, and can slow search dramatically.

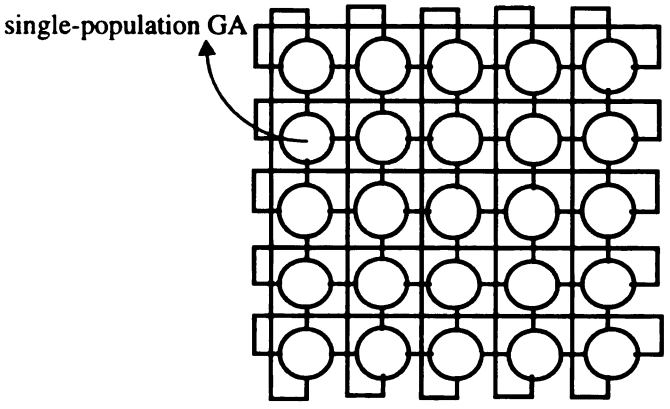
4.4 Hybrid PGA Models

Two hybrid PGA models have been developed and are presented here. One is an embedding of fgGAs into cgGAs. Figure 4.1 (a) shows an example in which each subpopulation on the ring is a fgGA with toroidal connections. There are two frequencies of migration in this model: frequency of migration on the ring and frequency of migration within the torus. The frequency of migration on the ring is much smaller than that within each torus. This model allows multiple fgGAs to evolve somewhat independently, with occasional interchange of solutions between these subpopulations.

The other hybrid model is a compromise between a cgGA and a fgGA. In this model, the connection topology used in the cgGA is one which is typically found in fgGAs, and a relatively large number of nodes is used. Figure 4.1 (b) shows an example in which each node of the torus is a single-population GA. The frequency of migration between neighbors is resembles that found in cgGAs. This model can be viewed as a special cgGA in which subpopulations are spatially arrayed in some fgGA connection topology.



(a) An example of embedding fgGAs into cgGAs



(b) An example of connecting cgGAs in a fgGA-style topology

Figure 4.1 Examples of the hybrid models

CHAPTER 5

Computational Study - Static JSSPs

The static GA-based scheduling system is evaluated on some benchmark JSSPs before the extended scheduling system is applied to dynamic JSSPs. First, the single-population GA is tested on two famous benchmark problems. This chapter then focuses on the investigation of the effect of parallelizing GAs and comparison of PGA models. The scheduling system with the best PGA model is also applied to other benchmark JSSPs.

5.1 The Effect of Parallelizing GAs

The scheduling system described in Chapter 3 has been implemented in GALOPPS (http://isl.cps.msu.edu/GA/) [63], a freeware GA development system from the MSU GARAGe, and run in a UNIX environment. As a benchmark, two FT problems, FT10x10 and FT20x5 [8], were tested. (The FT 6x6 problem is of less interest because it is relatively easy to obtain the optimum.) These two FT problems are of particular interest because almost all JSSP algorithms proposed have used them as benchmarks. Table 5.1 summarizes the best results obtained by previous approaches for the two FT problems. Except for the first three approaches, which are based on branch and bound methods, the remaining approaches are GA-based methods. In our experiments, the crossover and mutation rates were 0.6 and 0.1 respectively, and offspring replaced their parents, with

Table 5.1 Best Results obtained by previous approaches on the two FT problems

reference	10x10	20x5
Baker and McMahon (1985)[116]	960	1303
Adams *et al.* (1988)[94]	**930**	1178
Carlier and Pinson (1989) [115]	**930**	**1165**
Nakano and Yamada (1991)[35]	965	1215
Yamada and Nakano (1992)[36]	**930**	1184
Storer *et al.* (1993)[83]	954	1180
Dorndorf and Pesch (1993)[80]	**930**	**1165**
Fang *et al.* (1993)[40]	949	1189
Juels and Wattenberg (1994)[107]	937	1174
Mattfeld *et al.* (1994)[90]	**930**	**1165**
Dorndorf and Pesch (1995)[81]	938	1178
Bierwirth (1995)[91]	936	1181
Kobayashi *et al.* (1995)[82]	**930**	1173
Lin *et al.* (1996) [108]	**930**	**1165**

elitism protecting the best individual from replacement. The selection method was stochastic universal sampling [65]. The fitness scaling method was sigma truncation [64] with *sigma_trunc* = 2.0 followed by linear scaling [30] with *scalemult* = 1.3. For the FT10x10 problem, the average execution time of a single-population GA with population size 100 and 2000 generations is 110 sec. on a SUN Ultra 1. By using single-population GAs with our THX operators, we were able to find the global optima for the two problems, which are 930 and 1165, respectively. The FT10x10 was also used to evaluate the effectiveness of PGAs and to compare the performance of the various PGA models.

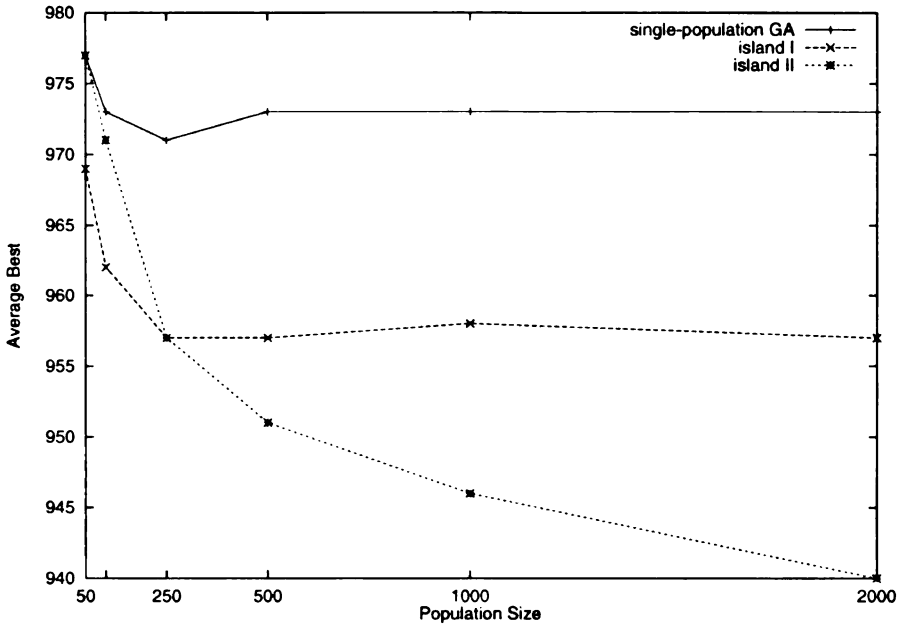
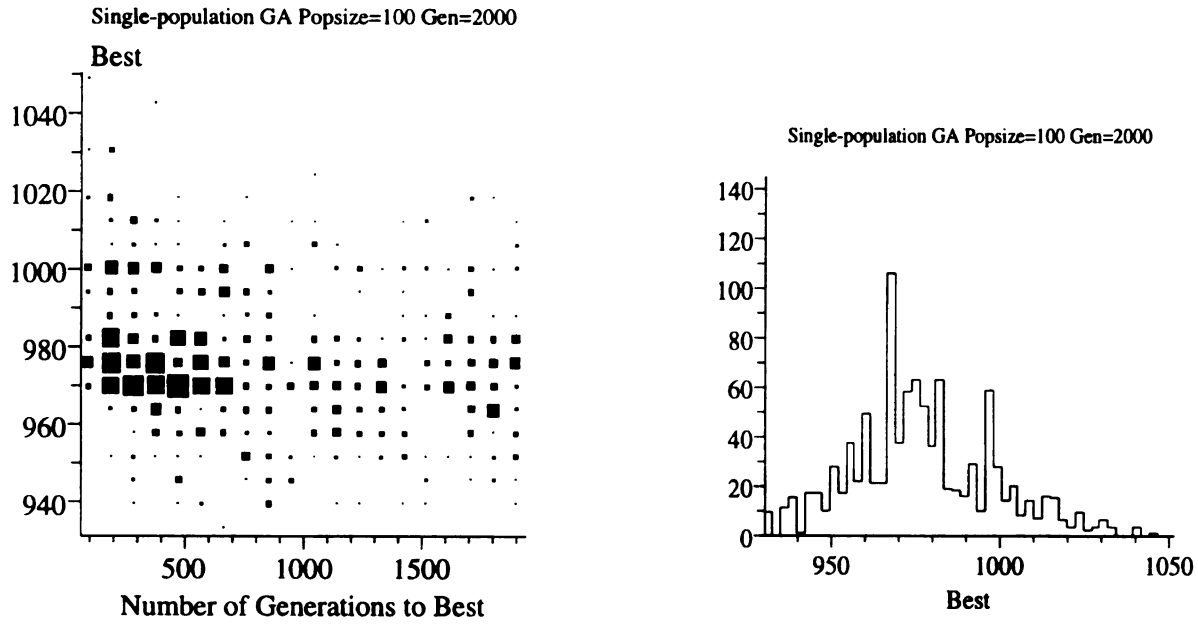


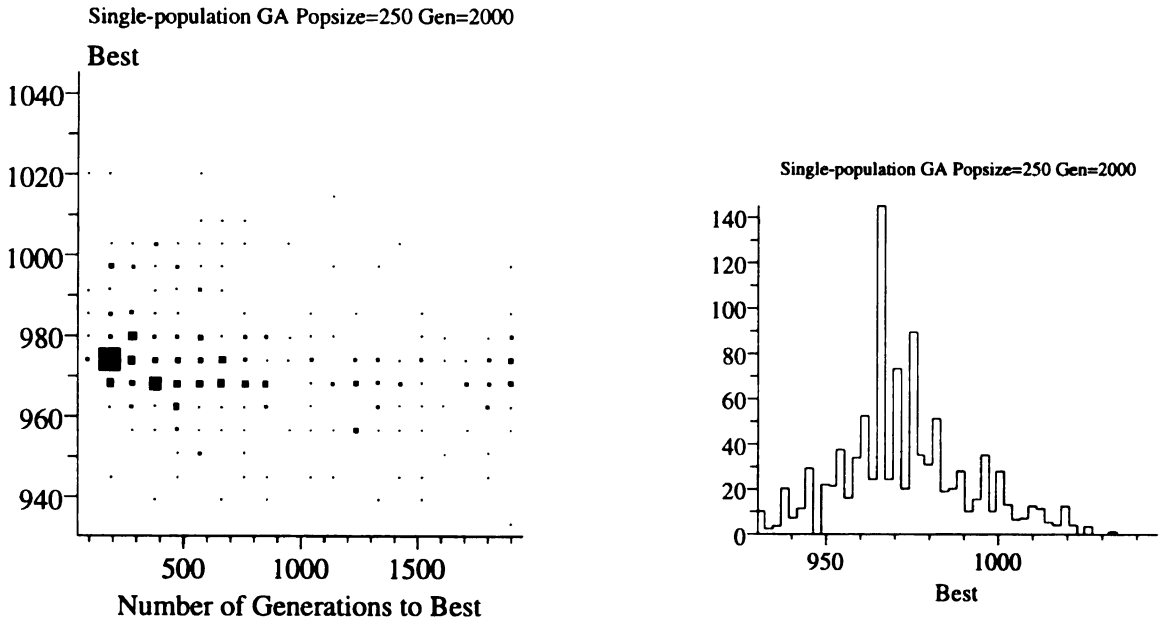
Figure 5.1 Average (100 runs) best results for three test models, various population sizes

To investigate the effect of parallelizing GAs, we used one single-population GA and two cgGAs with different population sizes on the FT10x10 problem. We varied the total population size in each case to test for its effect. The population sizes used were 50, 100, 250, 500, 1000, and 2000. Both cgGAs, called island I and island II, are connected in a one-way ring. The best individual is migrated to the next neighbor every 50 generations. The number of subpopulations in the island I GA was fixed at 5, so the subpopulation size is the total size divided by 5. In the island II GA, the subpopulation size is fixed at 50, so the number of subpopulations is obtained by dividing the total population size by 50. The number of generations for all runs is 2000.

Figure 5.1 shows the average best results of the three models on various population sizes based on 100 runs. The single-population GA doesn't show any improvement in performance after the population size 250 mark. In the single-population GA, the

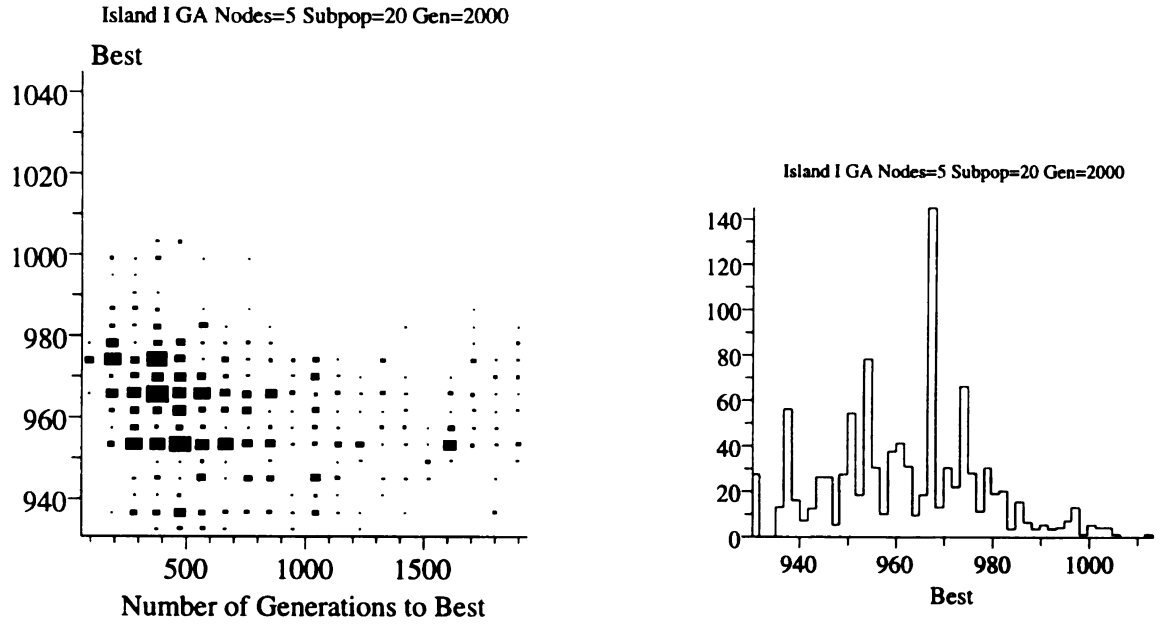


(a)

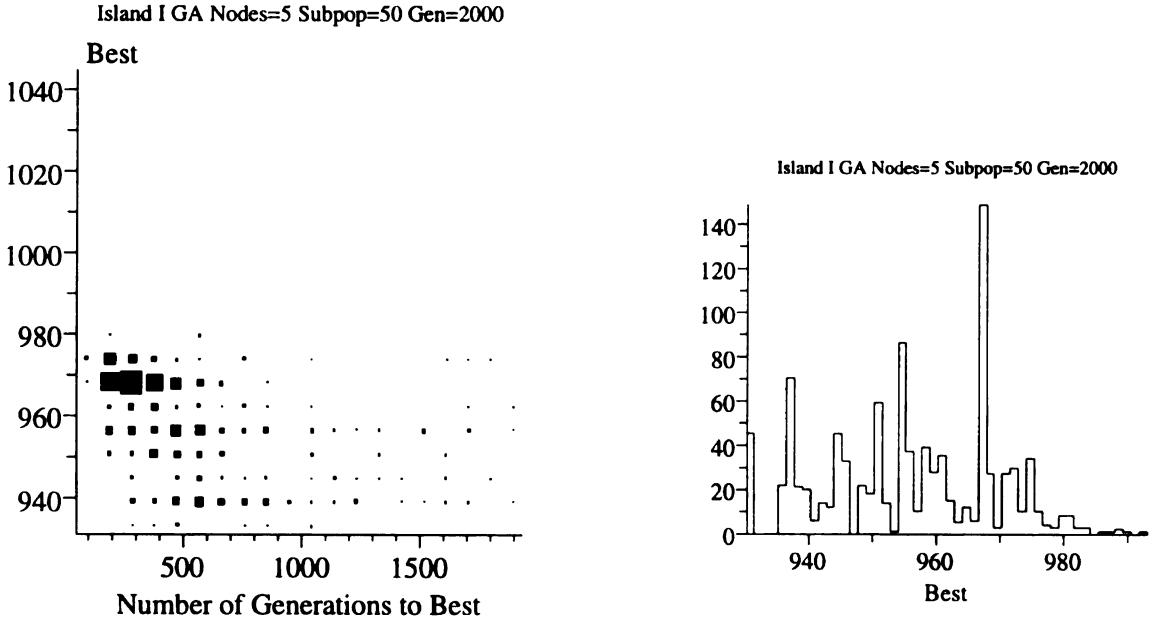


(b)

Figure 5.2 Single-population GAs with popsize 100 and 250



(a)



(b)

Figure 5.3 Island I GA with popsize 100 and 250

probability to have extraordinary individuals in the initial population is higher in larger populations. Therefore, if the selection pressure is high, the extraordinary individuals can quickly dominate the entire population and cause premature convergence. This might explain why there is no improvement in performance after the population size 250 mark in the single-population GA. Although the problem also appears in the island I GA model, it is alleviated by maintaining five subpopulations and the island I GA outperforms the single-population GA. The island II GA doesn't suffer as much from premature convergence. The larger the number of subpopulations, the better the diversity that is maintained. An average best of 940 is reached when the population size is 2000. By considering the average turnaround time of each subpopulation to calculate a fixed number of generations, we can analyze the speed-up of PGAs. In general, increasing the number of processors leads to approximately linear speed-up. For example, for a total population size fixed at 1000, the speed-up for runs with either 5 or 20 subpopulations are 4.7 and 18.5, respectively. We believe that the degraded performance is due to the communication overhead. In PGAs, we are more interested in the time needed to reach a given solution quality. Figure 5.2 shows the two-dimensional cell plot and the histogram of the single-population GA with population size 100 and 250, based on 1000 runs. The same plots of the island I GA is shown in Figure 5.3. In both figures, increasing the population size causes that the distribution of the results move to the left corner of the two-dimensional cell plots and to the left of the histogram. This indicates the improvement is made due to the increasing population size from 100 to 250. By comparing Figure 5.2 (b) and Figure 5.3 (b), in the two-dimensional cell plots, we can observe that the distribution of the results moves to the left corner in the island I GA. That is, the parallelization of the GA

yields better results using fewer evaluations. Actually, the average number of generations to obtain the best result in the island I GA is 732, compared to 852 for the single-population GA. Because the average best result of the island I GA is better than that of the single-population GA, the speed-up under "time-to-solution" is greater than 852 / (732/ 4.7) = 5.47.

5.2 Comparison of PGA models

We examined 5 PGA schemes -- the two cgGAs discussed in Section 5.1, plus one fgGA torus model and two hybrid models. (We didn't examine iiGAs because it is difficult to generate a meaningful but less detailed representation for the scheduling problem being addressed, such that less detailed individuals or schedules can be injected into subpopulations with more detailed representations while preserving their beneficial properties.) The migration interval for the cgGAs is 50 generations (*i.e.* an exchange between subpopulations every 50 generations). The population structures are shown in Table 5.2 in a subpopulation_size:connection_topology format. In the torus model, the subpopulation size is fixed at 2. In the hybrid I model, each island on the ring is a torus and the number of islands is fixed at 5. The number of generations for all runs is 2000. Figure

Table 5.2 The population structures of the PGA models

Popsize	Island I	Island II	torus	hybrid I	hybrid II
250	50:5	50:5	2:25x5	2:5 islands, each island:5x5 torus	10:5x5
500	100:5	50:10	2:25x10	2:5 islands, each island:10x5 torus	10:5x10
1000	200:5	50:20	2:25x20	2:5 islands, each island:10x10 torus	10:10x10
2000	400:5	50:40	2:25x40	2:5 islands, each island:10x20 torus	20:10x10

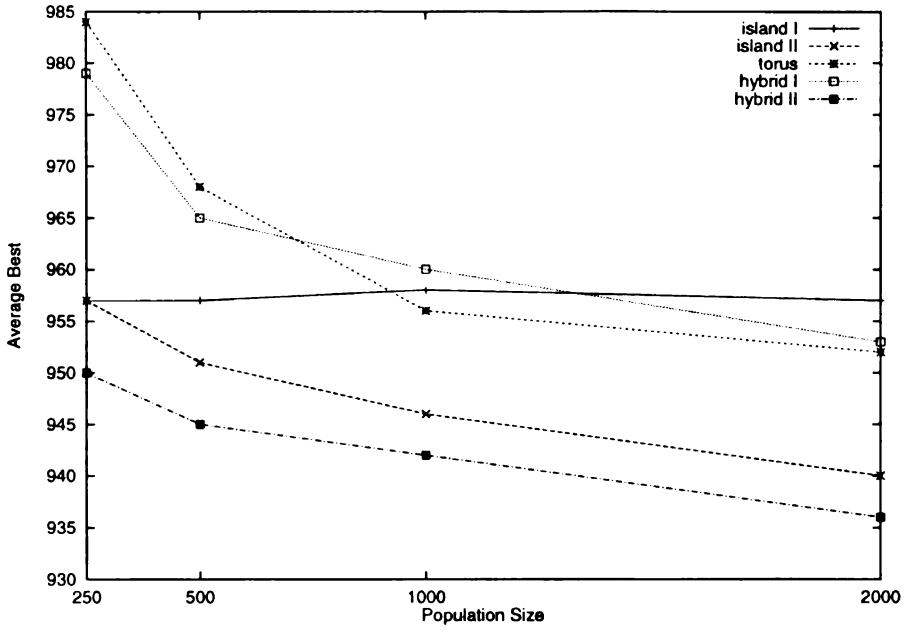


Figure 5.4 Average best of the five PGA models with various population sizes

5.4 shows the average best of the five PGA models based on 100 runs. The hybrid I and torus models have similar performance because both models are based on the fgGA model. Although both models are inferior to island I when the population size is less than 1000, their average best result improved for larger population sizes. The island II and hybrid II models are superior to the other approaches. The best performance is achieved by hybrid II model. Notice that in the hybrid II model at population size 2000, the optimal schedule is found 40 times in 100 runs and the *average* result is 936, which is within 0.7% of the optimum, and the standard deviation is 5.62. Because not all previous researchers reported their means and standard deviations, here we compare our best results with Juels and Wattenberg [107] and Mattfeld *et al.*[90]. Comparing with these two groups then, our PGA approach shows superior performance with a significance levels better than 0.0001.

Discussion: Finding a proper propagation speed of building blocks among

subpopulations is a major determinant of the performance of PGAs. With too rapid propagation, good individuals will quickly spread to other subpopulations and dominate the population. This results in premature convergence. If the propagation is too slow, PGAs will be close to multiple independent single-population GAs running at the same time and the advantage of interacting subpopulations in PGAs will vanish.

In fgGAs, if the population size is small, any strong local optima can be quickly propagated to the entire population, and fgGAs may perform worse than island GAs. That is why the torus and hybrid I models, which are based on the fgGA model, are inferior to island I when the population size is less than 1000. When the population size is large, these three models, -- torus, hybrid I, and island I -- will suffer from the problem of high selection pressure, as described in Section 5.1. Because the propagation speed in the torus and hybrid I models is less than in island I when the population size is large, both torus and hybrid I have better ability to maintain the diversity in the population and perform better than island I, as shown at population size 2000 in Figure 5.4. In the island II model, because of the fixed subpopulation size, the island II model doesn't suffer much from the problem when the population size is large. Besides, the longer ring supports slower propagation and allows more diversity to be maintained among subpopulations. That is why the island II model outperformed the torus, hybrid I, and island I models, as shown in Figure 5.4. In the hybrid II model, because the connection topology of fgGAs supports better propagation than in island I, the hybrid II model does better than the island II model, as shown in Figure 5.4.

In summary: fgGAs appear to lose genetic diversity too quickly, in comparison to cgGAs. Improvement can be made if a different migration strategy is applied [90]. In

Table 5.3 Computational results of the ABZ benchmarks

Problem	Size	SB I		SB II		Hybrid PGA		Best Known
		err.%	sec.[1]	err.%	sec.[1]	err.%	sec.[2]	
ABZ5	10x10	5.83	5.70	0.41	1503	0	128	1234
ABZ6	10x10	2.01	12.67	0	1101	0	133	943
ABZ7	20x15	9.77	118.87	6.77	1269	1.20	4356	665 [117]
ABZ8	20x15	15.52	125.02	6.87	1775	3.13	4036	670 [118]
ABZ9	20x15	9.48	94.32	7.14	1312	2.91	4278	686 [119]

1. The computing times were obtained on a VAX 780/11 [94].
2. The computing times were obtained on a SUN Ultra 1.

cgGAs, increasing the number of islands improves performance more than simply increasing the population size. Additionally, a good connection topology can further improve the propagation of building blocks and increase the performance. Best results were obtained with the hybrid model consisting of cgGAs connected in a fgGA-style topology.

5.3 Other Benchmark Results

Besides the well-known FT benchmark problems, there are many benchmark JSSPs available. The scheduling system with the best PGA model -- *i.e.*, the hybrid model consisting of cgGAs connected in a fgGA-style topology, was applied to three sets of benchmark JSSPs, which can be obtained from OR-Library (http://mscmga.ms.ic.ac.uk/info.html). In the following experiments of the hybrid PGA, the connection topology is a

Table 5.4 Computational results of the YN benchmarks*

Problem	Random		GA/GT		Single-pop. GA		Hybrid PGA		Best Known
	Average	Best	Average	Best	Average	Best	Average	Best	
YN1	52.79	26.80	10.25	8.90	5.29	2.70	3.49	2.36	888 [120]
YN2	44.58	21.05	4.50	3.62	4.25	3.07	3.29	2.85	912 [121]
YN3	46.27	23.27	6.57	5.90	5.34	2.78	3.12	1.56	898 [120]
YN4	44.76	23.03	8.50	7.68	5.94	3.17	3.17	2.66	977 [120]

* The results shown are the percentage of the relative error to the best known result.

10x10 torus and the subpopulation size on each node is 10. The number of generations for each run is 2000.

The first set is five instances prefixed with "ABZ," which were generated by Adams *et al.* [94]. The processing times were randomly drawn from a uniform distribution on the interval [50, 100] for ABZ5, [25, 100] for ABZ6, and [11, 40] for ABZ7-9. Adams *et al.* applied the shifting bottleneck algorithm to the five problems. SB I and SB II are the straight and the enumerative versions of the shifting bottleneck algorithm, respectively. Table 5.3 shows the computational results of SB I, SB II, and the hybrid PGA scheduling system. The relative error is calculated by *100*(best_result - best_known) / best_known.* The best results and the average computing times of the hybrid PGA were based on 100 runs. When the problem size is large, GAs found much better results than the other two methods, at additional computational cost.

The second set is four 20x20 instances prefixed with "YN," which were generated by

Yamada and Nakano [36] with processing times randomly drawn from a uniform distribution on the interval [10, 50]. As described in Chapter 2, Yamada and Nakano designed a GT-algorithm-based operator, GA/GT crossover. They applied this operator to the four JSSPs. Without showing the computation time, they compared the results of GA/GT with the randomly generated 400,000 active schedules for each problem. Table 5.4 shows the comparison of the hybrid PGA with the single-population GA, GA/GT, and randomly generated active schedules. The population size of the single-population GA is 250. The results shown are the percentage of the relative error to the best-known result. The average and best results of the hybrid PGA and the single-population GA are based on 100 runs. For the four problems, hybrid PGA, which yields averages within 4% and best within 3% of the best known, outperforms other methods. Compared to GA/GT, the better results obtained by the single-population GA also shows that THX crossover and mutation can transfer better information than GA/GT.

The last set is 10 10x10 instances prefixed with "ORB," which were generated by Applegate and Cook [109]. Applegate and Cook are able, through a combination of bottle, shuffle, and edge-finder, to solve the ten problems, but they did not report their run times. The bottle and shuffle are two heuristic methods for obtaining an initial schedule. The edge-finding algorithm is a branch and bound method. Table 5.5 shows the results obtained by the hybrid PGA. The hybrid PGA found the optimum in 9 out of 10 problems.

5.4 Summary

This chapter applied the GA scheduling system, based on the G&T algorithm, to static JSSPs. Our extensions to the G&T algorithm, the THX crossover and mutation operators, are designed to transmit the temporal relationships in the schedule. For both FT problems,

Table 5.5 Computational results of the ORB benchmarks

Problem	Best of 100 runs	Average	Time (sec.)	Optimum
ORB1	1059	1071.8	326	1059
ORB2	888	890.4	303	888
ORB3	1005	1024.5	401	1005
ORB4	1005	1016.2	365	1005
ORB5	889	892.8	222	887
ORB6	1010	1019.3	329	1010
ORB7	397	400.0	173	397
ORB8	899	910.7	358	899
ORB9	934	938.0	280	934
ORB10	944	945.2	290	944

the methods introduced found the optimum. The results show that although the specific operators are difficult to design, if problem-specific knowledge is successfully incorporated into the operators, the GA can work more effectively on the particular problem.

We further compared single-population GAs and PGAs on the FT10x10 problem. The effect of parallelizing the GA was twofold. PGAs not only alleviated the premature convergence problem and improve the results, but also found the solution in a fewer number of evaluations compared to single-population GAs. We also reported on various PGA models. In cgGAs, the number of islands used in the run had a greater positive effect on performance than simply increasing population size. In the fgGA model, premature convergence was still a problem, since the overlapping subpopulations are susceptible to domination by high-fitness individuals. Furthermore, the hybrid II model performed best due to the integration of the advantages of cgGAs and fgGAs.

In Section 5.3, we tested three sets of benchmark JSSPs. The impressive results show the effectiveness of the genetic operators and the hybrid PGA model. Although the scheduling system didn't find the best results known for some problems, the small relative errors show that near-optimal results were still obtained. From a practical viewpoint, instead of putting a large effort into finding the optimal schedule, producing a good schedule and maintaining this performance in a dynamic environment are the main concerns. Besides, the models described by static JSSPs are usually only crude representations of the actual problems, so the optimal schedule for the model may not lead to the best schedule for the actual problem. In the next chapter, we will apply the scheduling system to a more realistic model -- *i.e.*, dynamic JSSPs.

CHAPTER 6

Computational Study - Dynamic JSSPs

Manufacturing environments in the real world are subject to many sources of change which are typically treated as random occurrences, such as new job releases, machine breakdowns, job cancellations, due date changes, etc. Due to their dynamic nature, real world scheduling problems are rather computationally complex and are known to be strongly NP-hard [1]. This chapter considers dynamic JSSPs, in which jobs arrive continually. In practice, the most often used heuristic method is priority rules. As described in Section 2.3, when a machine becomes available, the priorities of the jobs currently available to the machine are calculated and the highest priority job is scheduled for work. General speaking, priority rules are computationally efficient and are useful for finding a reasonably good solution.

Instead of using priority rules, this chapter applies the GA-based scheduling system described in Chapter 3 to address dynamic JSSPs. In deterministic dynamic JSSPs, a set of benchmark problems is tested and the results are compared with another GA-based scheduling system and some priority rule-based systems. In stochastic JSSPs, a simulation model under various manufacturing environments is designed. The results are also compared with use of some priority rule-based systems.

6.1 Deterministic dynamic JSSPs

Deterministic dynamic JSSPs can be solved in either an exact or a heuristic manner. One example of an exact method is a depth-first branch-and-bound algorithm which builds a schedule forward in time [58]. Heuristic methods are especially interesting for practical applications. The most often used heuristic method is the priority rule approach, which schedules the highest priority job whenever a machine becomes available. The priority is based on some easily computed parameters of the jobs, operations, or machines, such as processing times, due dates, release times, and machine loadings. This section applies the GA-based scheduling system developed here to deterministic dynamic JSSPs, using a variety of objective functions. The results of priority rules and another GA-based scheduling system are also presented for comparison.

6.1.1 Experimental Design

The test set of 12 deterministic problems is taken from [53]. The characteristics of these problems are shown in Table 6.1. The tardiness factor measures approximately the proportion of tardy jobs. Due dates are then tightened or loosened to give the specific tardiness factor. For example, if the tardiness factor is set at 0.3, roughly 30% of the jobs are tardy. The due dates of the 30% jobs are then set tight, and the due dates of the remaining jobs are set loose. In this experimental design, the bottleneck machine is determined as the machine with the largest expected processing time.

The objective functions examined (Table 3.1) are weighted flow time, maximum tardiness, weighted tardiness, weighted lateness, weighted number of tardy jobs, and weighted earliness plus weighted tardiness. The definitions of the priority rules used for comparison in this chapter can be found in Table 2.3. In minimizing the weighted flow

Table 6.1 The characteristic of the test deterministic JSSPs

Problem	Size	Tardiness Factor	Utilization of Bottleneck Machine
JB1	10x3	0.4	0.6
JB2	10x3	0.2	0.6
JB4	10x5	0.7	0.6
JB9	15x3	0.7	0.9
JB11	15x5	0.2	0.9
JB12	15x5	0.4	0.6
LJB1	30x3	0.5	0.6
LJB2	30x3	0.5	0.75
LJB7	50x5	0.4	0.6
LJB9	50x5	0.7	0.9
LJB10	50x8	0.7	0.6
LJB12	50x8	0.7	0.9

time, we compared our GA approach with five priority rules -- RANDOM, FCFS, WSPT, WLWKR, and WTWORK. For the other objective functions, the priority rules used for comparison were WSPT, EGD, EOD, EMOD, MST, WS/OP, WCR, WCOVERT, and WR&M, except that in minimizing the maximum tardiness, which does not consider the weights of jobs, the non-weighted version of the priority rules, obtained by removing the weighted coefficient from the weighted version, was used. In addition to these priority rules, we also compared our results with Fang's results [59]. Fang's GA approach is described in Chapter 2.

In all runs, the crossover and mutation rates were 0.6 and 0.1 respectively, and offspring replaced their parents, with elitism protecting the best individual from replacement. Two versions of the GA were tested for the deterministic problems. One was a single-population GA with population size 50. The other was a hybrid PGA in which 25 single-population GAs with subpopulation size of 20 were connected in a 5x5 torus. The migration interval was 50 generations. The length of the run number for both versions was 50x(number of jobs). The best result obtained from 10 runs of the corresponding GA for each problem was recorded.

6.1.2 Experimental Results and Discussion

The results for each objective function are given below. The reported results are the normalized values (Table 3.1).

(1) Weighted Flow Time

Table 6.2 shows the results of priority rules. The best result for each problem is shaded in light gray. WSPT, which found the best result in 9 out of 12 problems, performed better than the other four priority rules. Figure 6.1 presents the comparison of the best results

Table 6.2 Priority rules for (normalized) weighted flow time in deterministic dynamic JSSPs

Problem	RND	FCSFS	WSPT	WLWKR	WTWORK
JB1	1.245	1.259	1.231	1.231	1.231
JB2	1.787	1.906	1.772	1.776	1.776
JB4	1.111	1.111	1.111	1.111	1.111
JB9	2.098	2.228	1.947	2.171	2.323
JB11	1.829	1.829	1.843	1.795	1.812
JB12	1.259	1.259	1.280	1.257	1.280
LJB1	1.509	1.509	1.494	1.497	1.504
LJB2	2.018	2.018	1.924	1.979	1.984
LJB7	1.800	1.761	1.692	1.812	1.828
LJB9	2.886	3.013	2.575	2.490	2.830
LJB10	1.885	1.898	1.776	1.853	1.947
LJB12	2.440	2.440	2.207	2.263	2.343

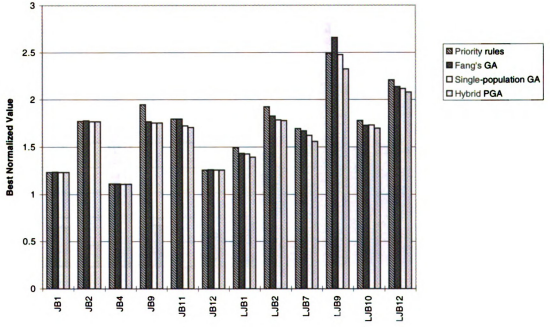


Figure 6.1 Comparison of priority rules and GA approaches for (normalized) weighted flow time in deterministic dynamic JSSPs

Table 6.3 Priority rules for (normalized) weighted tardiness in deterministic dynamic JSSPs

Problem	WSPT	EGD	EOD	EMOD	MST	WS/OP	WCR	WCOVERT	WR&M
JB1	0.178	0.220	0.220	0.220	0.193	0.193	0.193	0.178	0.178
JB2	0.194	0.086	0.086	0.086	0.086	0.086	0.123	0.087	0.087
JB4	0.560	0.560	0.560	0.560	0.560	0.560	0.560	0.560	0.560
JB9	0.530	0.279	0.269	0.276	0.294	0.406	0.383	0.289	0.185
JB11	0.087	0.019	0.042	0.060	0.018	0.018	0.023	0.008	0.000
JB12	0.283	0.218	0.221	0.221	0.218	0.218	0.218	0.223	0.218
LJB1	0.286	0.348	0.338	0.338	0.356	0.305	0.321	0.276	0.285
LJB2	0.588	0.469	0.460	0.470	0.478	0.466	0.463	0.499	0.464
LJB7	0.258	0.130	0.122	0.129	0.109	0.155	0.163	0.114	0.115
LJB9	0.936	1.290	1.301	1.162	1.192	1.061	0.998	0.796	0.843
LJB10	0.621	0.737	0.668	0.747	0.809	0.649	0.616	0.479	0.558
LJB12	0.712	0.544	0.565	0.603	0.658	0.704	0.702	0.489	0.524

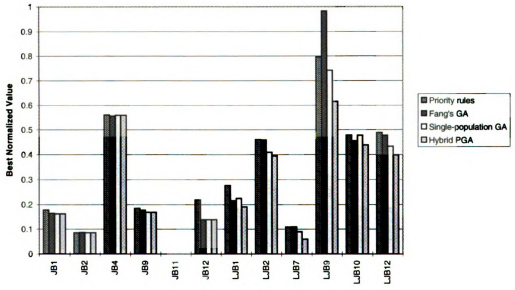


Figure 6.2 Comparison of priority rules and GA approaches for (normalized) weighted tardiness in deterministic dynamic JSSPs

Table 6.4 Priority rules for (normalized) maximum tardiness in deterministic dynamic JSSPs

Problem	SPT	EGD	EOD	EMOD	MST	S/OP	CR	COVERT	R&M
JB1	0.097	0.097	0.097	0.097	0.082	0.082	0.082	0.082	0.082
JB2	0.170	0.055	0.055	0.055	0.055	0.055	0.064	0.064	0.064
JB4	0.203	0.203	0.203	0.203	0.203	0.203	0.203	0.203	0.203
JB9	0.227	0.067	0.105	0.105	0.091	0.091	0.067	0.105	0.105
JB11	0.047	0.010	0.033	0.033	0.015	0.015	0.015	0.015	0.002
JB12	0.071	0.071	0.071	0.071	0.071	0.071	0.071	0.071	0.071
LJB1	0.056	0.065	0.065	0.075	0.065	0.065	0.065	0.081	0.081
LJB2	0.124	0.078	0.099	0.099	0.079	0.079	0.096	0.108	0.108
LJB7	0.022	0.036	0.036	0.036	0.031	0.031	0.031	0.036	0.031
LJB9	0.160	0.056	0.086	0.091	0.050	0.050	0.070	0.091	0.090
LJB10	0.051	0.043	0.052	0.073	0.050	0.050	0.051	0.068	0.068
LJB12	0.076	0.035	0.041	0.041	0.037	0.037	0.037	0.098	0.098

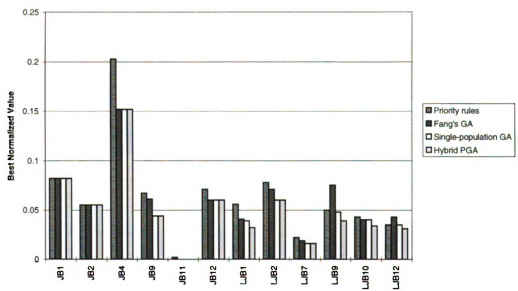


Figure 6.3 Comparison of priority rules and GA approaches for (normalized) maximum tardiness in deterministic dynamic JSSPs

Table 6.5 Priority rules for (normalized) weighted lateness in deterministic dynamic JSSPs

Problem	WSPT	EGD	EOD	EMOD	MST	WS/OP	WCR	WCOVERT	WR&M
JB1	-0.173	-0.144	-0.144	-0.144	-0.158	-0.158	-0.158	-0.173	-0.173
JB2	-0.812	-0.730	-0.730	-0.730	-0.730	-0.730	-0.677	-0.809	-0.809
JB4	0.497	0.497	0.497	0.497	0.497	0.497	0.497	0.497	0.497
JB9	0.115	0.159	0.142	0.156	0.174	0.290	0.268	0.016	-0.047
JB11	-0.607	-0.413	-0.417	-0.418	-0.391	-0.379	-0.376	-0.542	-0.587
JB12	-0.079	-0.069	-0.082	-0.082	-0.069	-0.069	-0.069	-0.047	-0.069
LJB1	-0.112	-0.016	-0.061	-0.065	-0.007	-0.059	-0.042	-0.103	-0.109
LJB2	0.070	0.042	0.018	0.028	0.052	0.040	0.030	0.131	0.013
LJB7	-0.373	-0.259	-0.282	-0.278	-0.298	-0.259	-0.272	-0.411	-0.338
LJB9	0.622	1.173	1.185	1.048	1.090	0.944	0.889	0.657	0.720
LJB10	-0.038	0.275	0.188	0.196	0.385	0.280	0.183	0.076	0.166
LJB12	0.256	0.385	0.380	0.401	0.545	0.564	0.566	0.312	0.363

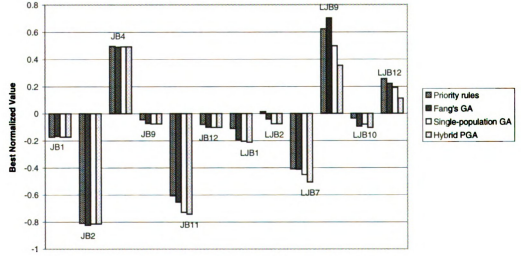


Figure 6.4 Comparison of priority rules and GA approaches for (normalized) weighted lateness in deterministic dynamic JSSPs

Table 6.6 Priority rules for (normalized) weighted number of tardy jobs in deterministic dynamic JSSPs

Problem	WSPT	EGD	EOD	EMOD	MST	WS/OP	WCR	WCOVERT	WR&M
JB1	0.433	0.457	0.457	0.457	0.524	0.524	0.524	0.433	0.433
JB2	0.118	0.233	0.233	0.233	0.233	0.233	0.294	0.097	0.097
JB4	0.709	0.709	0.709	0.709	0.709	0.709	0.709	0.709	0.709
JB9	0.425	0.754	0.718	0.754	0.754	0.837	0.837	0.483	0.483
JB11	0.269	0.254	0.136	0.136	0.224	0.224	0.255	0.129	0.000
JB12	0.516	0.431	0.451	0.451	0.431	0.431	0.431	0.431	0.431
LJB1	0.401	0.519	0.519	0.451	0.519	0.519	0.519	0.401	0.426
LJB2	0.413	0.459	0.447	0.447	0.459	0.459	0.459	0.374	0.470
LJB7	0.383	0.492	0.431	0.431	0.294	0.368	0.365	0.311	0.348
LJB9	0.564	0.849	0.802	0.794	0.799	0.821	0.770	0.735	0.687
LJB10	0.533	0.672	0.644	0.638	0.695	0.729	0.684	0.610	0.627
LJB12	0.478	0.752	0.726	0.740	0.770	0.870	0.873	0.664	0.776

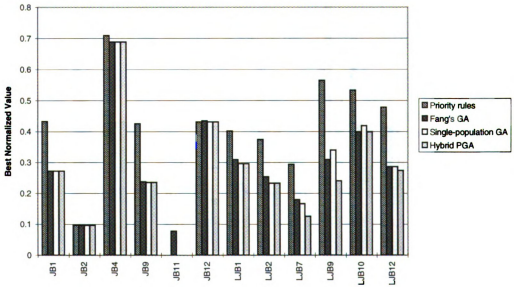


Figure 6.5 Comparison of priority rules and GA approaches for (normalized) weighted number of tardy jobs in deterministic dynamic JSSPs

Table 6.7 Priority rules for (normalized) weighted earliness plus weighted tardiness in deterministic dynamic JSSPs

Problem	WSPT	EGD	EOD	EMOD	MST	WS/OP	WCR	WCOVERT	WR&M
JB1	0.529	0.583	0.583	0.583	0.544	0.544	0.544	0.529	0.529
JB2	1.201	0.901	0.901	0.901	0.901	0.901	0.923	0.984	0.984
JB4	0.622	0.622	0.622	0.622	0.622	0.622	0.622	0.622	0.622
JB9	0.946	0.399	0.395	0.396	0.414	0.522	0.499	0.562	0.418
JB11	0.780	0.450	0.500	0.538	0.428	0.415	0.422	0.557	0.587
JB12	0.644	0.505	0.524	0.524	0.505	0.505	0.505	0.494	0.505
LJB1	0.684	0.711	0.737	0.742	0.720	0.668	0.685	0.654	0.678
LJB2	1.106	0.895	0.901	0.911	0.904	0.893	0.895	0.868	0.915
LJB7	0.890	0.520	0.527	0.536	0.515	0.569	0.599	0.639	0.568
LJB9	1.250	1.408	1.417	1.277	1.293	1.178	1.106	0.935	0.965
LJB10	1.280	1.199	1.148	1.299	1.233	1.018	1.048	0.882	0.950
LJB12	1.168	0.702	0.750	0.805	0.770	0.845	0.839	0.667	0.684

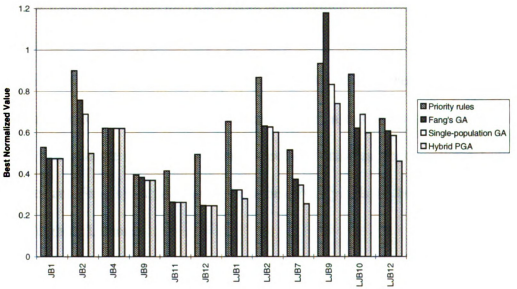


Figure 6.6 Comparison of priority rules and GA approaches for (normalized) weighted earliness plus weighted tardiness in deterministic dynamic JSSPs

obtained by priority rules and GA approaches. In general, priority rules and GA approaches show similar performance in small problems such as JB1, JB2, and JB4. Among the GA approaches, Fang's GA found better results than priority rules in 8 problems. Our single-population GA dominated priority rules and outperformed Fang's GA (only for LJB10 did Fang's GA find a better result). Furthermore, the hybrid PGA found the best results of any of the methods on all problems.

(2) Weighted Tardiness

Table 6.3 gives the results of priority rules. WCOVERT, which found the best results in 6 problems, shows better performance than other priority rules. Compared to GA approaches (Figure 6.2), both the single-population GA and the hybrid PGA found results at least as good as those obtained by priority rules. Among the GA approaches, Fang's GA found better results than the single-population GA in only 3 problems (JB4, LJB1, and LJB10). The hybrid GA yields the best overall performance.

(3) Maximum Tardiness

Table 6.4 shows the results of priority rules. EGD yields the best performance. The comparison of priority rules and GA approaches is shown in Figure 6.3. Both the single-population GA and the hybrid PGA found results at least as good as those found by priority rules and Fang's GA. Moreover, the hybrid PGA yields the best performance.

(4) Weighted Lateness

The results of priority rules are shown in Table 6.5. Basically, minimizing weighted flow time and minimizing weighted lateness are equivalent. The best priority rule in the objective of minimizing weighted flow time, WSPT, also gave the best performance among priority rules here. Figure 6.4 presents the comparison of priority rules and GA

approaches. Both the single-population GA and the hybrid PGA dominated priority rules. Among the GA approaches, single-population GA found better results than Fang's GA in 9 problems. The hybrid PGA found better results than Fang's GA in 11 problems and found results at least as good as those obtained by the single-population GA. Therefore, the hybrid PGA shows the best performance.

(5) Weighted Number of Tardy Jobs

Table 6.6 presents the results of priority rules with the objective of minimizing weighted number of tardy jobs. Among the priority rules, WSPT and WCOVERT show better performance than other priority rules. Figure 6.5 shows the comparison of priority rules and GA approaches. The GA approaches outperform the priority rules in most problems. Compared to priority rules, a great improvement is made by GA approaches in 8 problems (JB1, JB9, and all LJB problems). Among GA approaches, the hybrid PGA yields the best overall performance.

(6) Weighted Earliness plus Weighted Tardiness

Table 6.7 presents the results of priority rules. WCOVERT, which found the best results in 8 problems, gave the best performance among priority rules. Figure 6.6 shows the comparison of priority rules and GA approaches. The single-population GA yields results better than or equal to Fang's results in 10 problems. Only in JB4, Fang's GA found a better result than the hybrid PGA.

In summary: our single-population GA and hybrid PGA both performed consistently better than the priority rules. The single-population GA yields results better than or equal to Fang's in 61 of 72 scenarios. The hybrid PGA is seen to provide the best results. Only 3 of its 72 scenarios are worse than Fang's results. The average percentage of improvement

Table 6.8 The average percentage of improvement over the best found by priority rules and Fang's GA and the corresponding level of significance α

Objective function	Single-population GA		Hybrid PGA	
	Improvement %	α	Improvement %	α
Weighted Flow Time	1.03	0.0093	2.46	0.0048
Weighted Tardiness	3.35	0.063	9.88	0.022
Maximum Tardiness	5.67	0.045	10.80	0.0054
Weighted Lateness	10.85	0.030	19.63	0.024
Weighted Number of Tardy Jobs	0.46	0.15	5.79	0.029
Weighted Earliness plus Weighted Tardiness	2.18	0.12	11.51	0.010

over the best found by priority rules and Fang's GA with respect to various objective functions is shown in Table 6.8. The relative improvement of the hybrid PGA is larger for weighted tardiness, maximum tardiness, weighted lateness, and weighted earliness plus weighted tardiness. Table 6.8 also shows the corresponding level of significance α for a one-tailed test concerning paired differences. Both GA approaches retain their effectiveness for all objective functions. As shown in the table, all results hold at significance levels of 0.15 or bette, and all PGA improvements are significant at the 3% level or better. The superior results show that the THX crossover and mutation successfully transmit useful characteristics -- *i.e.*, the temporal relationships among operations. Furthermore, the hybrid PGA performs better than the single-population GA because the premature convergence problem is alleviated by parallelizing the GA,

allowing better global search.

6.2 Stochastic dynamic JSSPs

To study the effectiveness of the scheduling system in stochastic dynamic JSSPs, a simulation model of JSSPs provides an easy way to predict the performance and compare several alternatives under a wide variety of environments. The due date tightness and machine utilization are the main factors in the design of the simulation model. Another consideration is that the relative machine utilizations are unlikely to be equal in a manufacturing environment.

In this section, the simulation model is presented. Then, the results of the scheduling system are compared with those of priority rules with respect to different objective functions and manufacturing environments. Furthermore, the proposed rescheduling method is compared with rescheduling from scratch, for the objective of minimizing weighted flow time.

6.2.1 Experimental Design

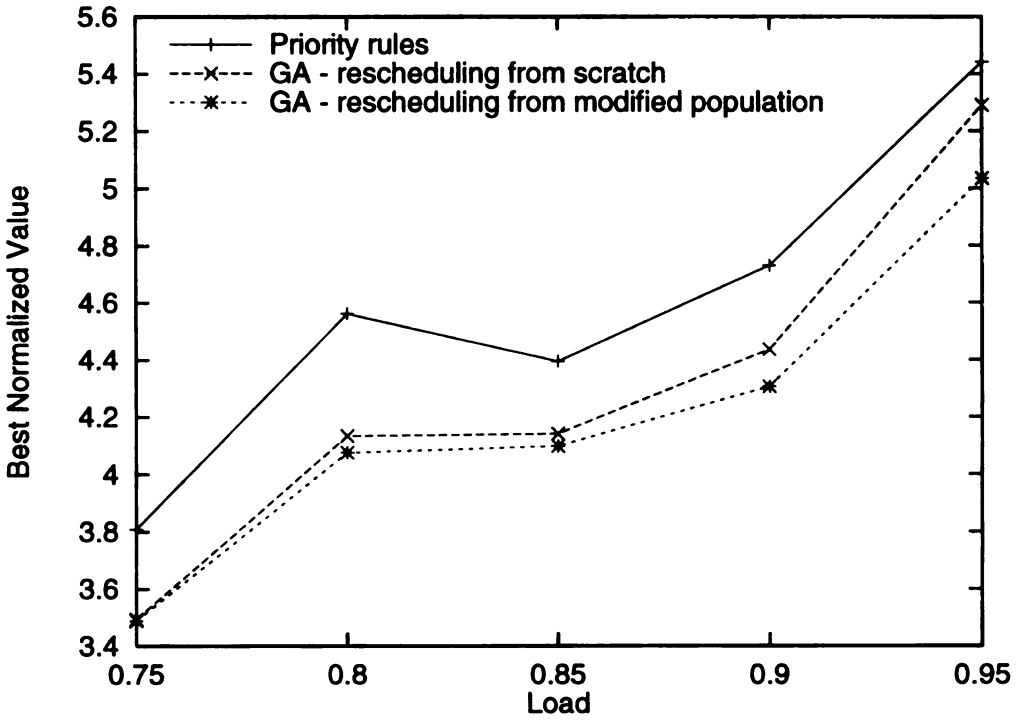
The stochastic job shop simulated has 5 machines with jobs arriving continually according to a Poisson process. The process is observed until the completion of 100 jobs. Each job has a random routing through the system. The operation processing times at each machine are uniformly distributed with various means to yield different levels of machine workload. Two classes of problems were designed. One was a balanced workload, with five levels of average machine utilization -- 75%, 80%, 85%, 90%, and 95%. The other was an unbalanced workload, with five levels of average machine utilization -- 60%, 65%, 70%, 75%, and 80%, in a 3:2 ratio of machine loads. The weights of jobs were uniformly

distributed between 1 and 2. For the objective of weighted flow time, no due date was assigned to the 10 scenarios, and 10 test problems were randomly generated for each scenario. In total, therefore, 100 problems were created for the objective of weighted flow time. For the other due-date-related objective functions, jobs have due dates set at arrival time plus F times their processing times, where F is the flow allowance factor to control due date tightness. Five levels of due date tightness were tested -- F = 2, 3, 4, 5, and 6. Therefore, there were 50 scenarios. For each scenario, 5 problems were randomly generated, so 250 problems were created in total for each due-date-related objective function.

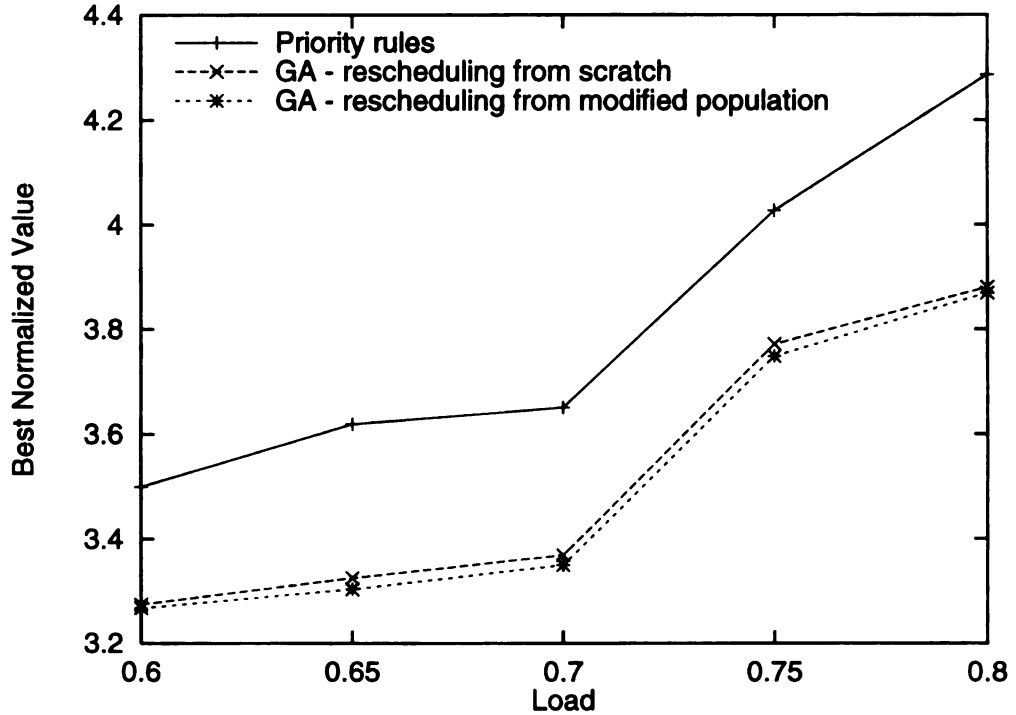
The priority rules for comparison were the same as in the study of deterministic problems. In the stochastic problems, a deterministic problem is generated whenever an event occurs -- *e.g.*, new job(s) arrive. The number of generations was set at 200 for each event. The average computational cost for each event is 7.0 seconds. For the objective of weighted flow time, two versions of the single-population GA were examined. The first reschedules the new deterministic problem from scratch. The second reschedules by using a modified population as described in Section 3.2. For the other objective functions, we applied only the second single-population GA to compare with the priority rules.

6.2.2 Experimental Results and Discussion

The results of the simulation experiment for each objective function are presented below. The reported results are the normalized values. The results of the two genetic approaches for the objective of weighted flow time are the average best results on the ten problems of each scenario, and the best results were obtained from 10 runs for each problem. For other objective functions, the results given are the average best results of the



(a) Weighted flow time vs. load, balanced workload



(b) Weighted flow time vs. load, unbalanced workload

Figure 6.7 Comparison of priority rules and GA approaches for (normalized) weighted flow time in stochastic dynamic JSSPs

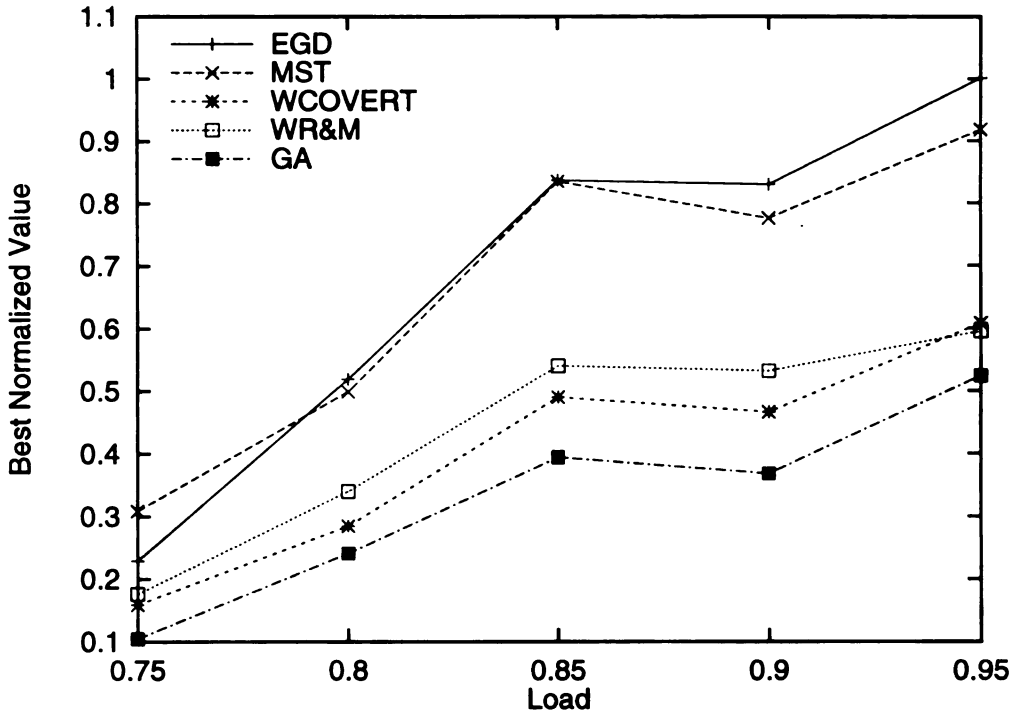
5 problems of each scenario, and the best results were obtained from 5 GA runs for each problem.

(1) Weighted Flow Time

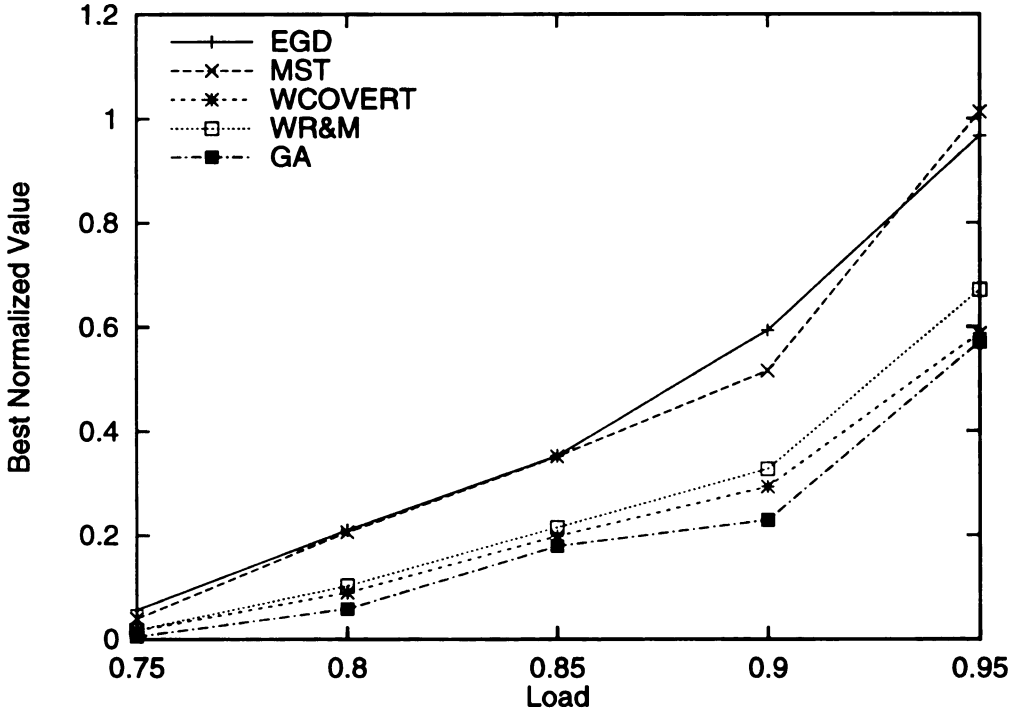
In the priority rules, WSPT is dominant at all utilization levels of balanced and unbalanced workload. Figure 6.7 shows the comparison of the best results of priority rules and GA approaches. The two GA approaches consistently found better results than priority rules and yielded comparable or better results at utilization levels less than 90%. The relative superiority of the GA with rescheduling from the modified population is higher under heavy workload, such as the runs of the 90% and 95% balanced workload models. This agrees with the implications of Section 3.2. Under heavy workload, jobs arrive closely after each other. Because only a few operations are removed from the system, most information retained in the last population before the new jobs arrive is still useful for the new problem. The modification process successfully preserves the information of the temporal relationships and enhances the efficiency of genetic search.

(2) Weighted Tardiness

The best four priority rules in the simulation are WCOVERT, WR&M, EGD, and MST. The results of others which have been consistently dominated by these four priority rules are omitted for greater clarity in presentation. In general, the problems become easier when the due dates are set loosely. While the GA dominates the priority rules at all flow allowances, the differences at flow allowance of 4, 5, and 6 are marginal. Figure 6.8 and Figure 6.9 show the comparison of the GA and the four priority rules at flow allowances of 2 and 3 under balanced and unbalanced workload. The GA is superior to the priority rules. The difference between the GA and the priority rules is increasingly significant with a

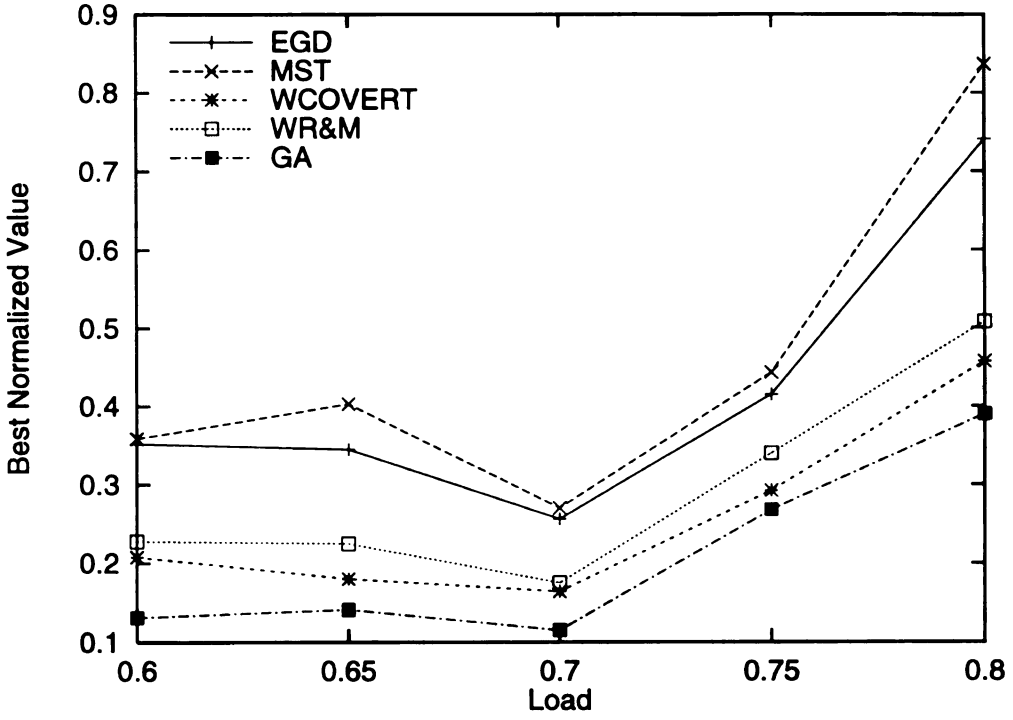


(a) Weighted tardiness vs. load at F=2, balanced workload

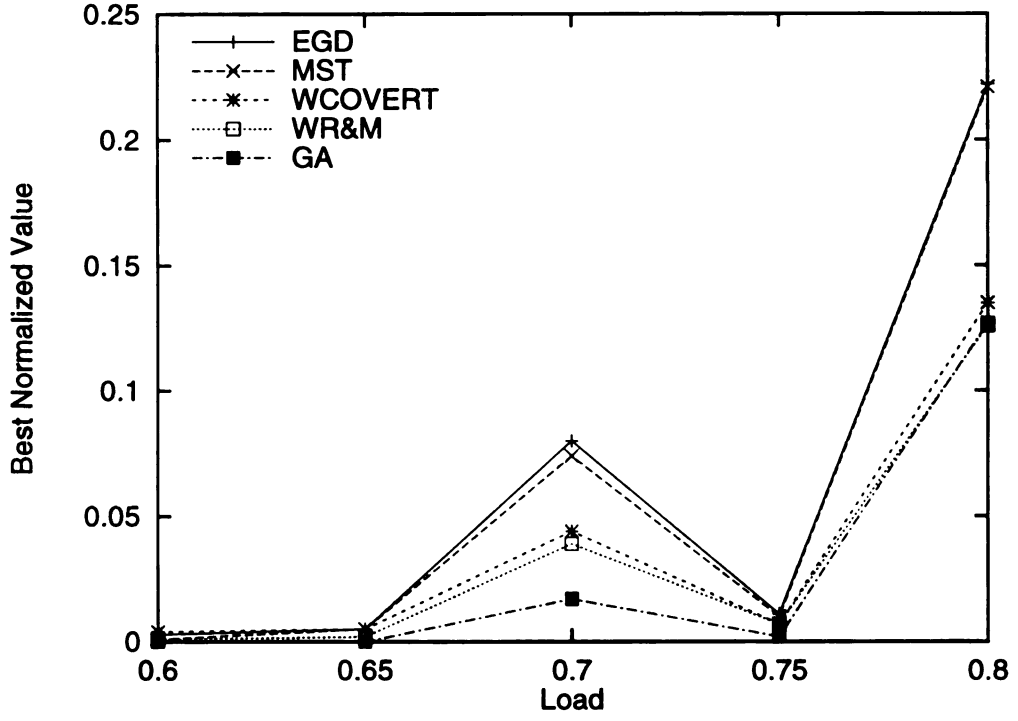


(b) Weighted tardiness vs. load at F=3, balanced workload

Figure 6.8 Comparison of priority rules and the GA approach for (normalized) weighted tardiness in stochastic dynamic JSSPs under balanced workload

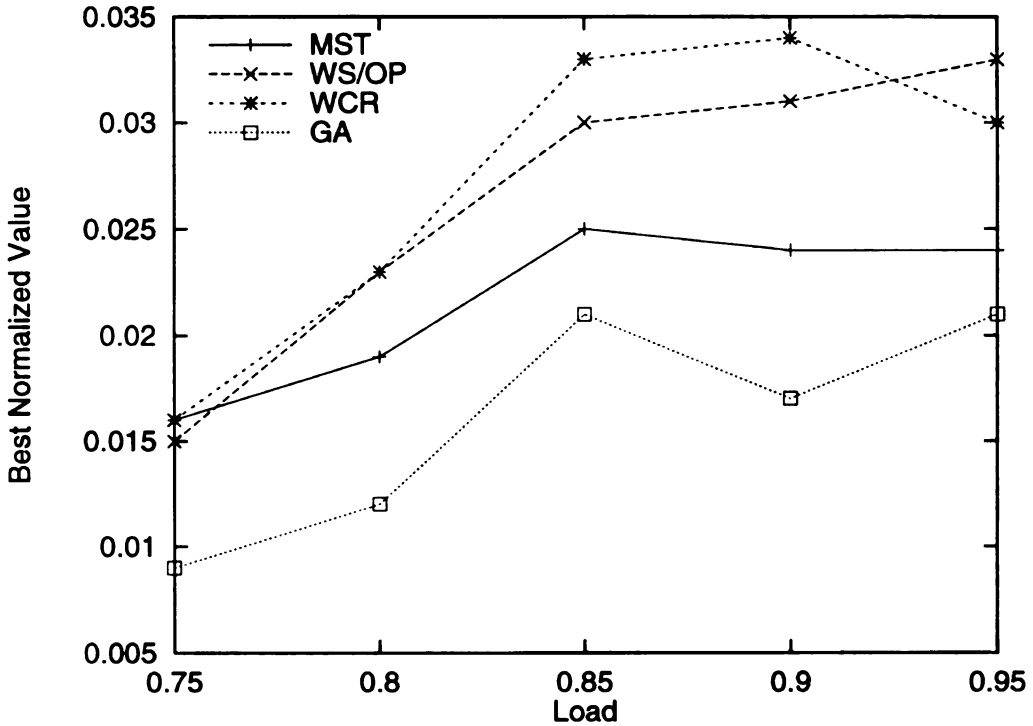


(a) Weighted tardiness vs. load at F=2, unbalanced workload

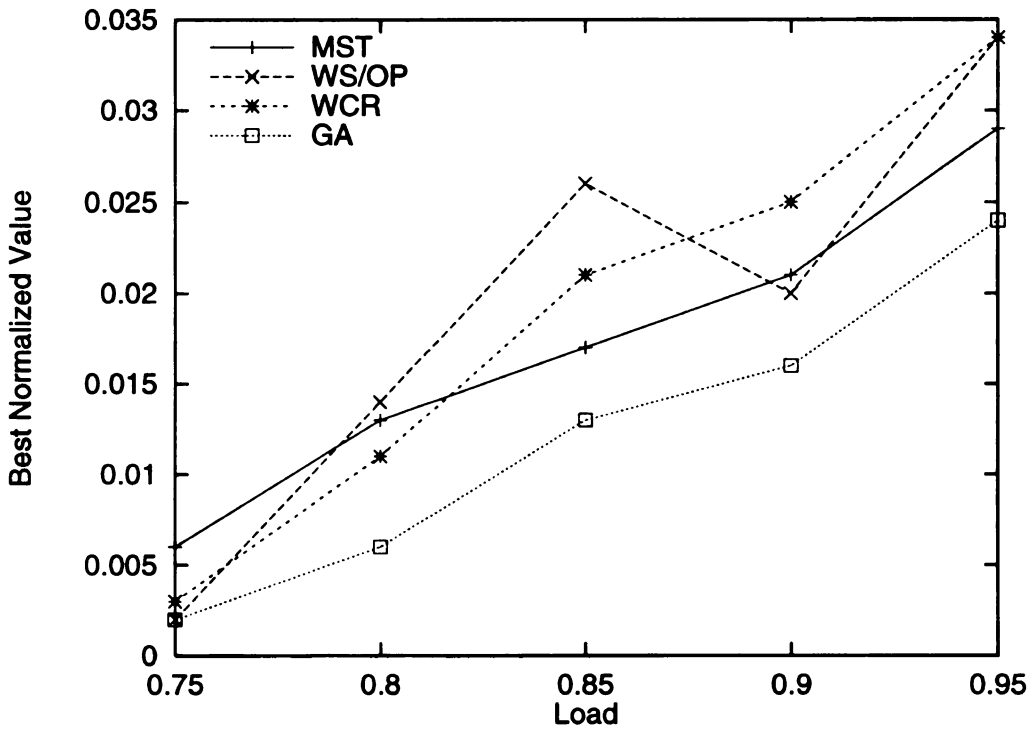


(b) Weighted tardiness vs. load at F=3, unbalanced workload

Figure 6.9 Comparison of priority rules and the GA approach for (normalized) weighted tardiness in stochastic dynamic JSSPs under unbalanced workload

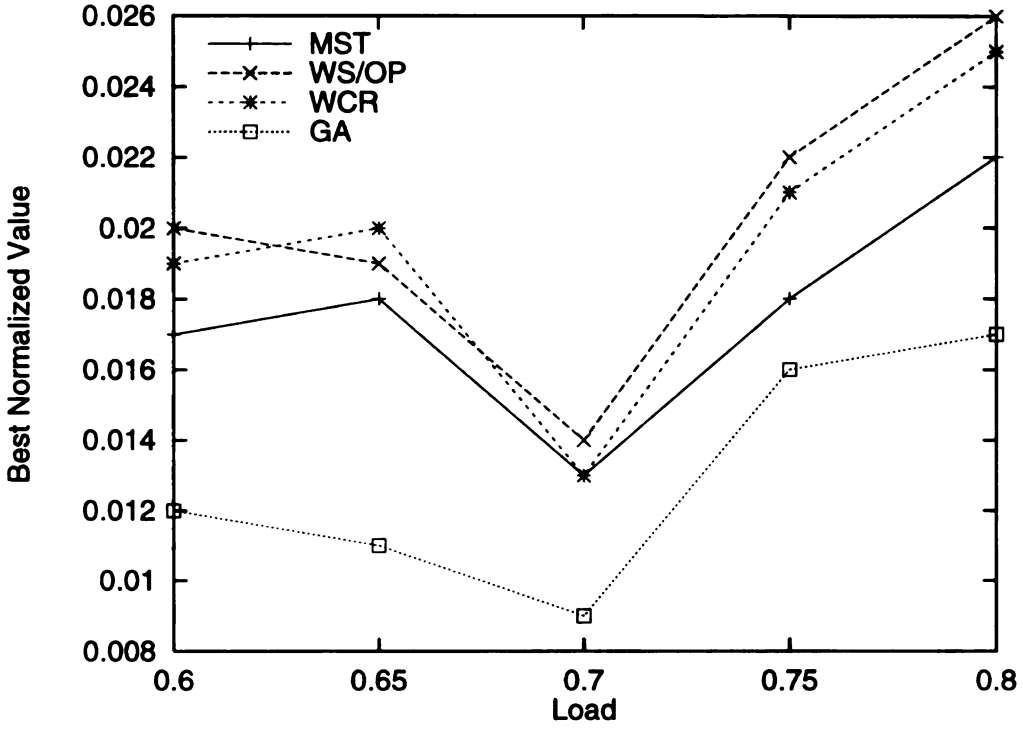


(a) Maximum tardiness vs. load at F=2, balanced workload

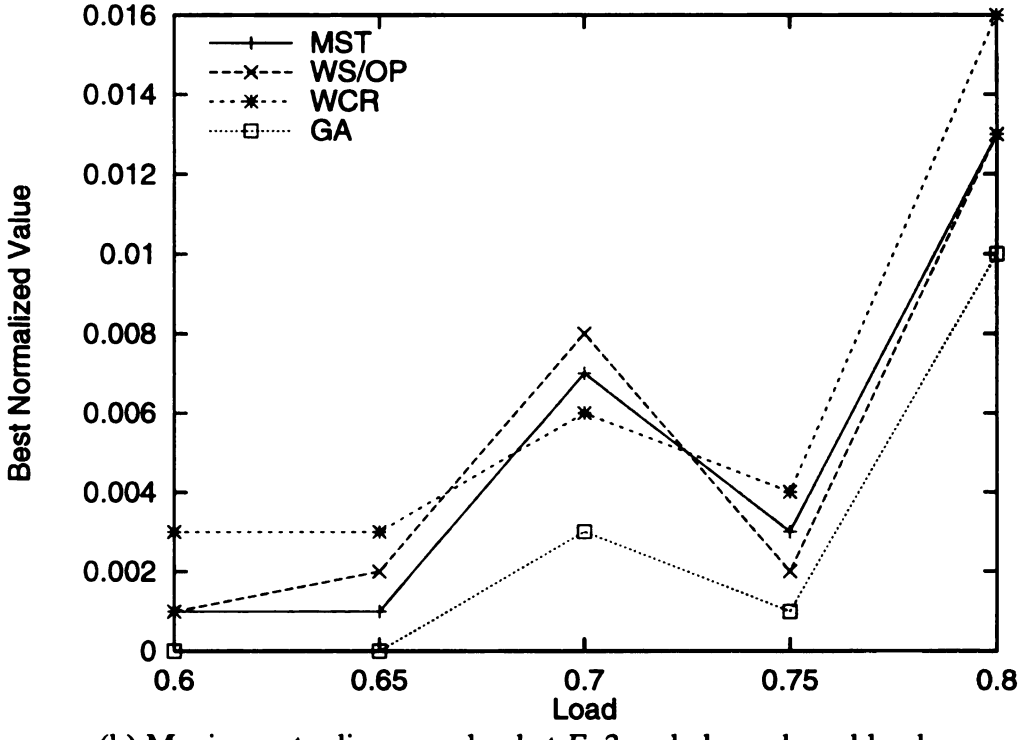


(b) Maximum tardiness vs. load at F=3, balanced workload

Figure 6.10 Comparison of priority rules and the GA approach for (normalized) maximum tardiness in stochastic dynamic JSSPs under balanced workload

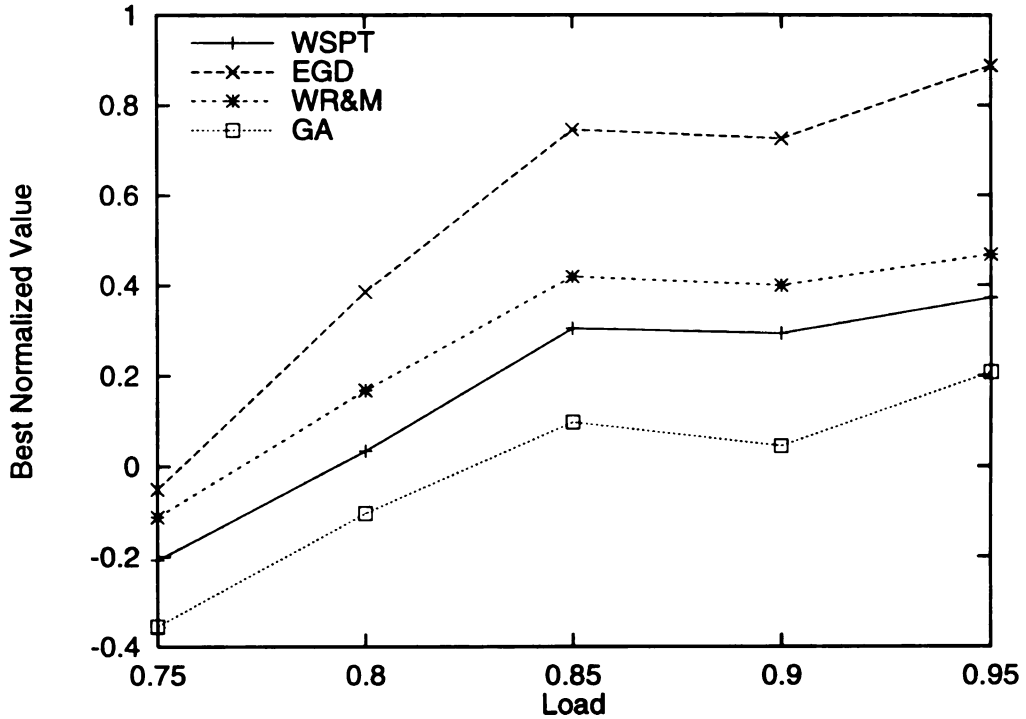


(a) Maximum tardiness vs. load at F=2, unbalanced workload

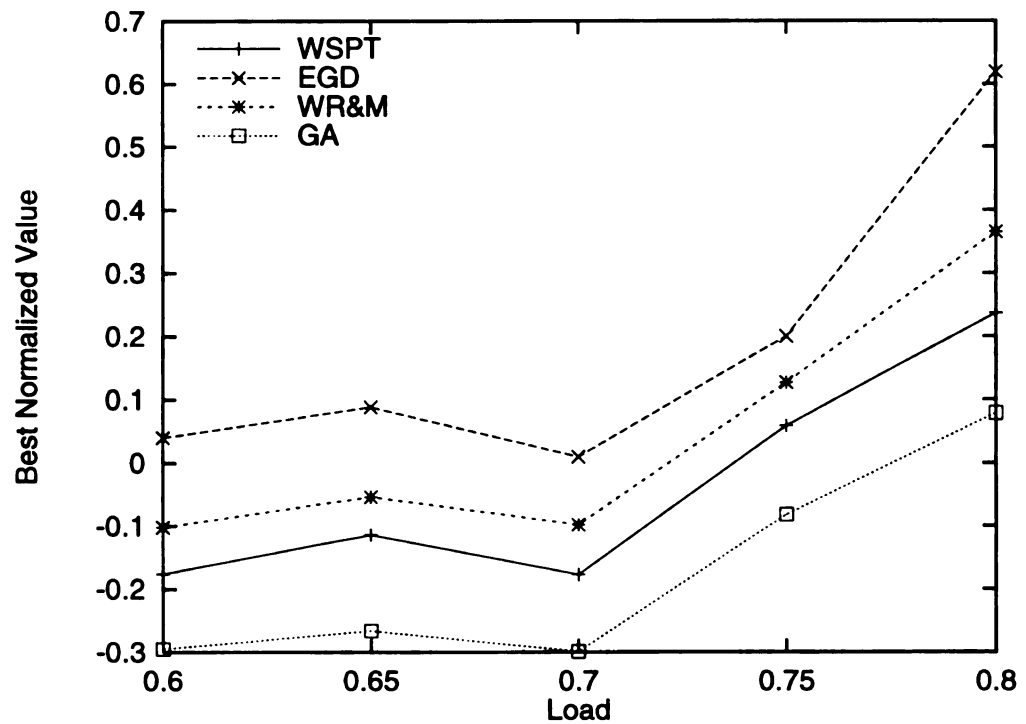


(b) Maximum tardiness vs. load at F=3, unbalanced workload

Figure 6.11 Comparison of priority rules and the GA approach for (normalized) maximum tardiness in stochastic dynamic JSSPs under unbalanced workload



(a) Weighted lateness vs. load at F=2, balanced workload



(b) Weighted lateness vs. load at F=2, unbalanced workload

Figure 6.12 Comparison of priority rules and the GA approach for (normalized) weighted lateness in stochastic dynamic JSSPs

decrease in the flow allowance. At flow allowance of 2, the difference is at the 0.0013 level of significance for balanced workload and at the 0.0029 level of significance for unbalanced workload.

(3) Maximum Tardiness

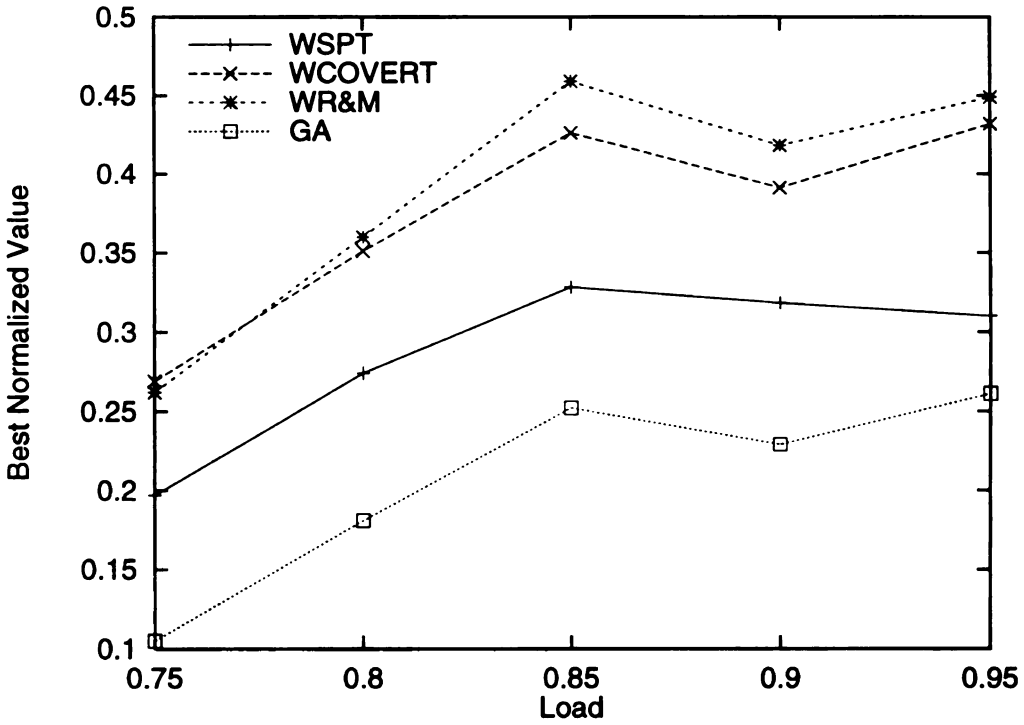
The GA is superior to the priority rules more significantly when the due dates are tight. Figure 6.10 and Figure 6.11 depict the comparison of the GA and the best three priority rules -- MST, WS/OP, and WCR, at flow allowances of 2 and 3 under balanced and unbalanced workloads. The GA retains superiority at significance levels of 0.05 or better for the case of balanced workload at flow allowances of 2, 3, 4, and 5, and at the significance levels of 0.15 for the case of unbalanced workload at flow allowances of 2, 3, and 4.

(4) Weighted Lateness

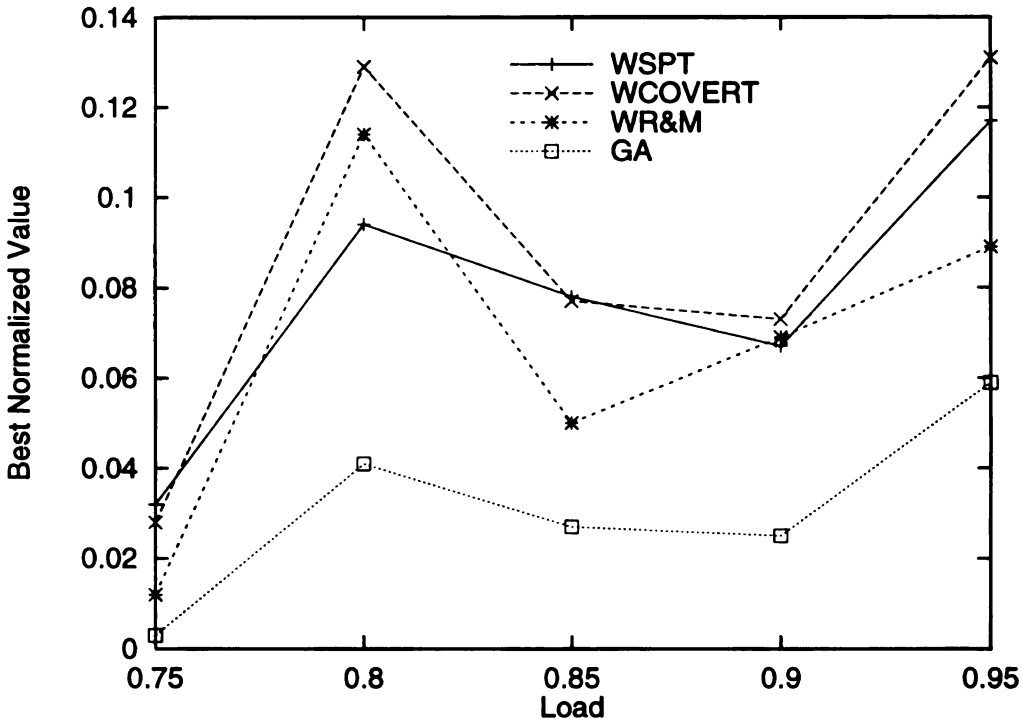
Consistent with the results of weighted flow time, WSPT dominates other priority rules at all flow allowances and load levels under balanced and unbalanced workloads. Figure 6.12 (a) and (b) show typical graphs of the comparison of the GA and the best three priority rules -- WSPT, EGD, and WR&M under balanced and unbalanced workloads, respectively. Unlike the results from previous objective functions, the GA performed consistently better than all priority rules at all flow allowances. The superiority is retained at significance levels of 0.0005 or better for the case of balanced workload, and 0.0039 or better for the case of unbalanced workload.

(5) Weighted Number of Tardy Jobs

The best three priority rules are WSPT, WCOVERT, and WR&M. When the flow allowance is 5 or 6, the difference between the GA and the priority rules is marginal.

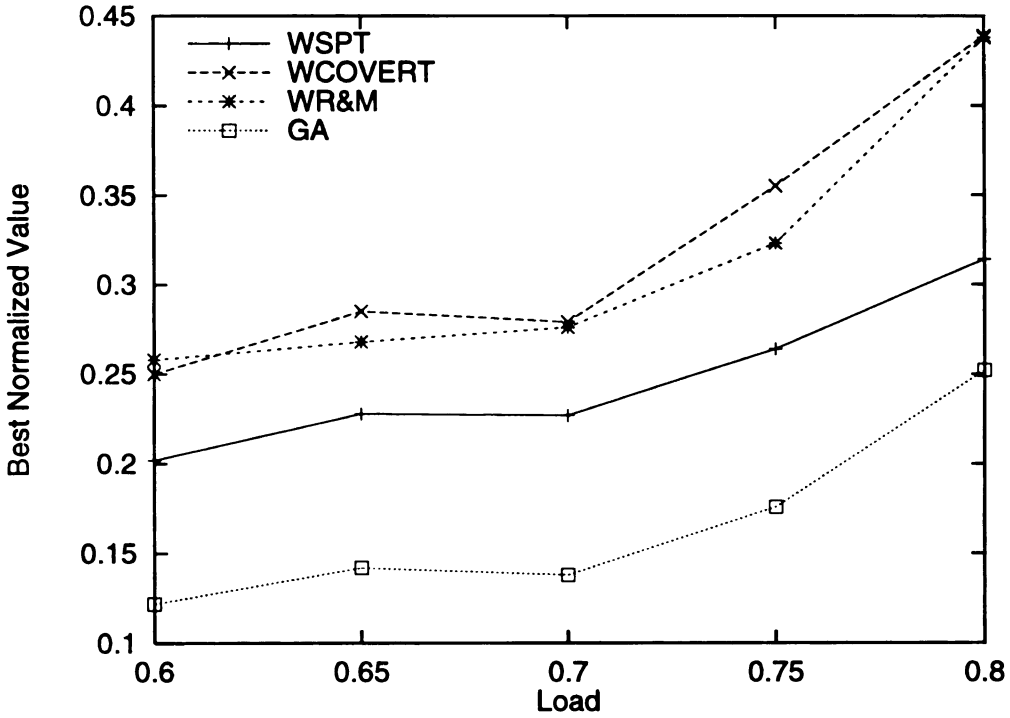


(a) Weighted number of tardy jobs vs. load at F=2, balanced workload

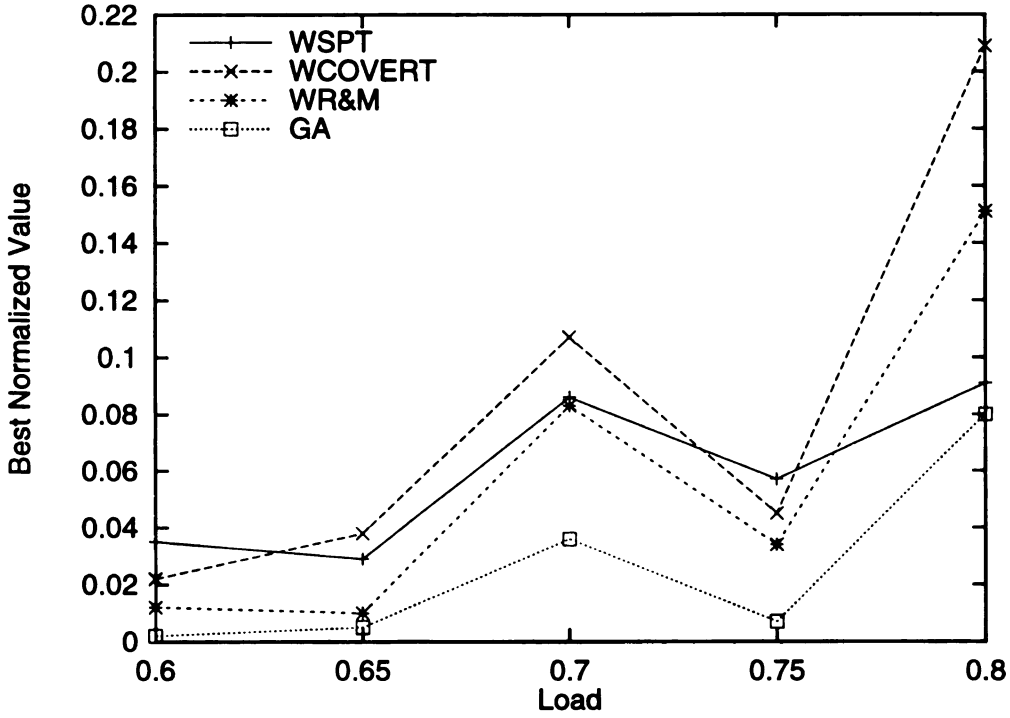


(b) Weighted number of tardy jobs vs. load at F=4, balanced workload

Figure 6.13 Comparison of priority rules and the GA approach for (normalized) weighted number of tardy jobs in stochastic JSSPs under balanced workload

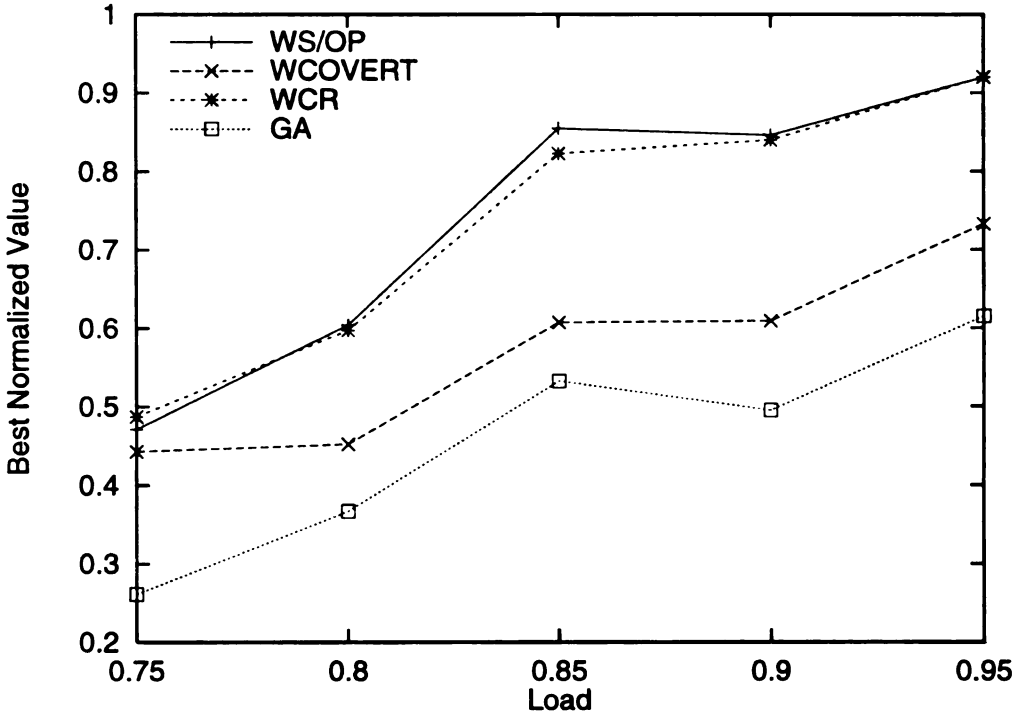


(a) Weighted number of tardy jobs vs. load at F=2, unbalanced workload

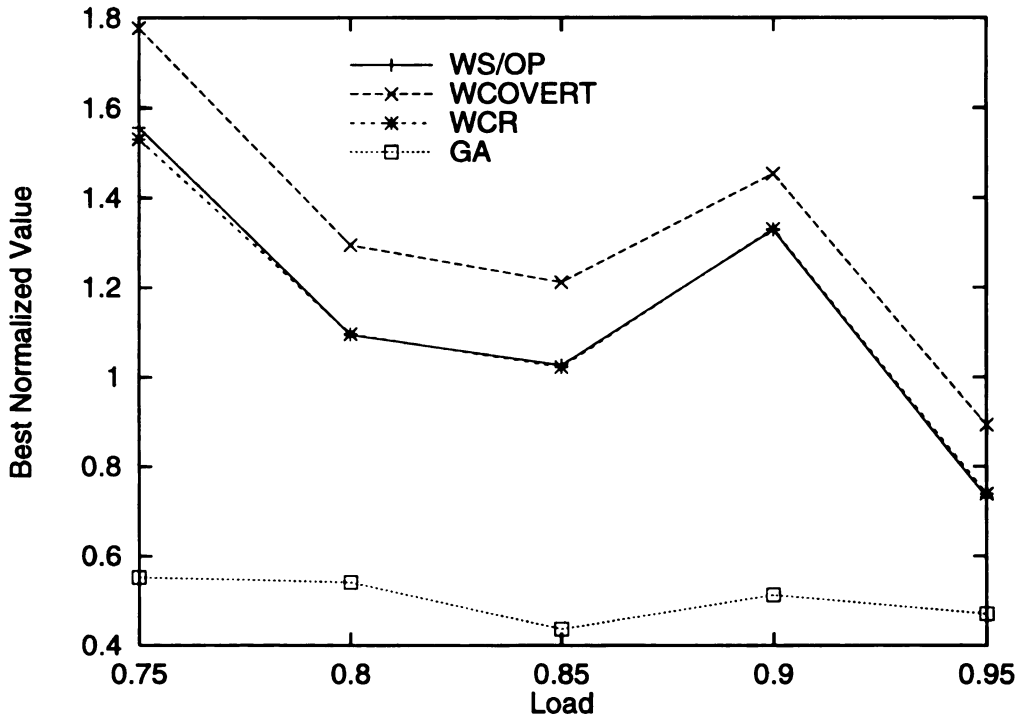


(b) Weighted number of tardy jobs vs. load at F=3, unbalanced workload

Figure 6.14 Comparison of priority rules and the GA approach for (normalized) weighted number of tardy jobs in stochastic JSSPs under unbalanced workload

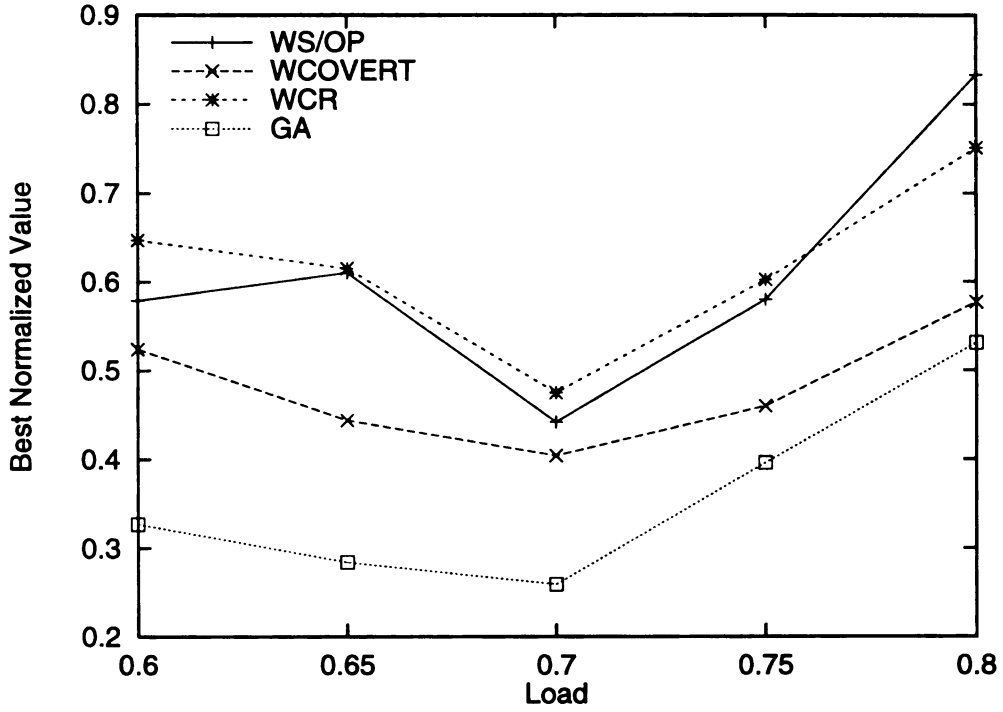


(a) Weighted earliness and weighted tardiness vs. load at F=2, balanced workload

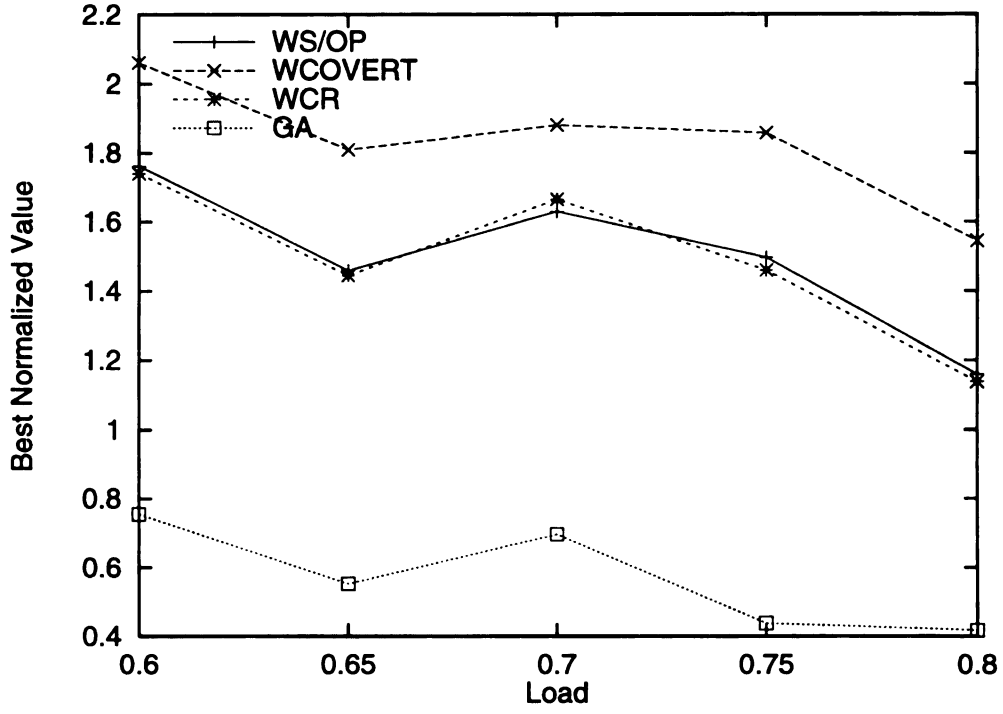


(a) Weighted earliness and weighted tardiness vs. load at F=4, balanced workload

Figure 6.15 Comparison of priority rules and the GA approach for (normalized) weighted earliness plus weighted tardiness in stochastic JSSPs under balanced workload



(a) Weighted earliness and weighted tardiness vs. load at F=2, unbalanced workload



(a) Weighted earliness and weighted tardiness vs. load at F=4, unbalanced workload

Figure 6.16 Comparison of priority rules and the GA approach for (normalized) weighted earliness and weighted tardiness in stochastic JSSPs under unbalanced workload

Figure 6.13 shows the comparison of the GA and the three priority rules at flow allowances of 2 and 4 under balanced workloads. The difference between the GA and the priority rules is more significant at tight due dates. The levels of significance are 0.003 and 0.0072 for flow allowances of 2 and 4, respectively. Notice also that WSPT dominates other rules when the due dates are tight. At flow allowances of 4, 5, and 6, WR&M yields better results than WSPT. The same findings are also shown in the case of unbalanced workload (Figure 6.14).

(6) Weighted Earliness plus Weighted Tardiness

The GA dominates the priority rules across all flow allowances and load levels. The relative improvement of the GA over the priority rules is retained at a significance level of 0.056 or better for the case of balanced workload and 0.0067 or better for the case of unbalanced workload. Figure 6.15 and Figure 6.16 show the comparison of the GA and the best three priority rules -- WS/OP, WCR, and WCOVERT, at flow allowances of 2 and 4 under balanced and unbalanced workloads. As seen from the figures, WCR and WS/OP have similar performance. WCOVERT is the best priority rule at flow allowance of 2. However, at flow allowance of 4, WCR and WS/OP outperform WCOVERT. Furthermore, unlike the other (regular) objective functions, the values decrease when the load level increases at flow allowances of 4, 5, and 6.

In summary: From the results of the objective function of weighted flow time, the proposed rescheduling method successfully preserves the information of the temporal relationships and enhances the efficiency of genetic search. Table 6.9 shows the results of all objective functions with respect to different workloads and due date tightnesses. We report only the results of the GA with rescheduling from a modified population and the

Table 6.9 The normalized results of the GA and the percentage improvement over the priority rules

F	Balanced Workload					Unbalanced Workload				
	75%	80%	85%	90%	95%	60%	65%	70%	75%	80%
Weighted Flow Time										
N/A	3.488(8.4)	4.076(10.7)	4.098(6.7)	4.305(9.0)	5.135(5.6)	3.268(6.6)	3.304(8.7)	3.350(8.2)	3.748(6.9)	3.869(9.7)
Weighted Tardiness										
2	0.105(34.0)	0.241(15.4)	0.394(19.6)	0.368(21.0)	0.524(11.9)	0.131(37.0)	0.141(21.7)	0.115(29.9)	0.268(8.2)	0.390(14.7)
3	0.005(70.6)	0.059(34.4)	0.179 (9.6)	0.229(21.8)	0.570 (2.9)	0.00*0.001	0.00*0.002	0.017(56.4)	0.002(71.4)	0.127 (-0.8)
4	0.001(80.0)	0.068(29.9)	0.008(57.9)	0.112 (-24)	0.097(11.8)	0.004 (0.0)	0.000 (0.0)	0.00*0.001	0.000 (0.0)	0.00*0.001
5	0.000 (0.0)	0.000 (0.0)	0.001 (0.0)	0.006(25.0)	0.056(6.7)	0.000 (0.0)	0.000 (0.0)	0.000 (0.0)	0.001(50.0)	0.000 (0.0)
6	0.000 (0.0)	0.000 (0.0)	0.000 (0.0)	0.000 (0.0)	0.000 (0.0)	0.000 (0.0)	0.000 (0.0)	0.000 (0.0)	0.000 (0.0)	0.000 (0.0)
Maximum Tardiness										
2	0.009(40.0)	0.012(36.8)	0.021(16.0)	0.017(29.2)	0.021(12.5)	0.012(29.4)	0.011(35.3)	0.009(30.8)	0.016(11.1)	0.017(22.7)
3	0.002 (0.0)	0.006(45.5)	0.013(23.5)	0.016(20.0)	0.024(17.2)	0.00*0.001	0.00*0.001	0.003(50.0)	0.001(50.0)	0.010(23.1)
4	0.001 (0.0)	0.006(33.3)	0.004(33.3)	0.006 (0.0)	0.008 (-14)	0.000 (0.0)	0.000 (0.0)	0.000 (0.0)	0.000 (0.0)	0.00*0.001
5	0.000 (0.0)	0.000 (0.0)	0.00*0.001	0.002(33.3)	0.005(28.6)	0.000 (0.0)	0.000 (0.0)	0.000 (0.0)	0.000 (0.0)	0.000 (0.0)
6	0.000 (0.0)	0.000 (0.0)	0.000 (0.0)	0.000 (0.0)	0.000 (0.0)	0.000 (0.0)	0.000 (0.0)	0.000 (0.0)	0.000 (0.0)	0.000 (0.0)
Weighted Lateness										
2	-0.36 (70.7)	-0.10(406)	0.098(67.8)	0.045(84.6)	0.208(44.1)	-0.29 (67.6)	-0.27 (133)	-0.29(68.9)	-0.08(239)	0.079(66.7)
3	-1.24(23.6)	-0.96(28.0)	-0.74(32.6)	-0.52(64.6)	-0.01(107)	-1.417(7.9)	-1.339(5.7)	-1.140 (14)	-1.36(13.3)	-0.871(5.4)
4	-2.251(8.3)	-1.64(11.8)	-1.72(13.6)	-1.84(10.0)	-1.34(12.8)	-2.415(4.0)	-2.259(6.8)	-2.296(4.9)	-2.318(6.4)	-1.99(11.0)
5	-3.261(6.6)	-3.091(5.2)	-2.835(7.1)	-2.674(7.2)	-2.232(5.3)	-3.535(2.5)	-3.334(3.7)	-3.375(4.1)	-3.235(5.6)	-3.038(5.8)
6	-3.863(4.9)	-4.062(4.7)	-3.735(3.9)	-3.898(5.1)	-3.295(7.4)	-4.431(2.2)	-4.472(3.4)	-4.245(3.1)	-3.940(4.8)	-3.957(4.1)
Weighted Number of Tardy Jobs										
2	0.105(46.7)	0.181(33.9)	0.252(23.2)	0.229(28.0)	0.261(15.8)	0.122(39.6)	0.142(37.7)	0.138(39.2)	0.176(33.3)	0.252(19.7)
3	0.017(68.5)	0.065(49.6)	0.089(36.0)	0.128(31.9)	0.187(21.4)	0.002(60.0)	0.005(50.0)	0.036(56.6)	0.007(77.4)	0.080(12.1)
4	0.003(75.0)	0.041(56.4)	0.027(46.0)	0.025(62.7)	0.059(33.7)	0.000 (0.0)	0.000 (0.0)	0.003 (0.0)	0.000 (0.0)	0.006 (-50)
5	0.000 (0.0)	0.000 (0.0)	0.007 (-17)	0.020 (4.8)	0.055(22.5)	0.000 (0.0)	0.000 (0.0)	0.000 (0.0)	0.007 (0.0)	0.000 (0.0)
6	0.000 (0.0)	0.000 (0.0)	0.000 (0.0)	0.003 (0.0)	0.004 (0.0)	0.000 (0.0)	0.000 (0.0)	0.000 (0.0)	0.000 (0.0)	0.000 (0.0)
Weighted Earliness plus Weighted Tardiness										
2	0.261(41.1)	0.367(18.8)	0.532(12.4)	0.495(18.7)	0.615(14.9)	0.327(37.6)	0.284(36.0)	0.259(35.9)	0.396(13.9)	0.531 (8.0)
3	0.260(63.7)	0.330(46.1)	0.450(31.7)	0.564 (7.2)	0.873 (8.2)	0.388(59.6)	0.262(68.8)	0.338(54.3)	0.309(64.4)	0.400(40.8)
4	0.552(63.9)	0.541(50.5)	0.437(57.2)	0.513(61.4)	0.471(35.6)	0.756(56.6)	0.553(61.7)	0.697(57.2)	0.438(70.0)	0.418(63.3)
5	1.018(55.8)	0.749(62.3)	0.665(62.9)	0.633(60.6)	0.711(48.7)	1.749(42.2)	1.371(47.1)	1.110(58.5)	1.076(55.1)	0.682(64.5)
6	1.097(57.4)	1.203(55.9)	0.927(61.9)	1.018(62.0)	0.699(55.9)	2.378(36.7)	2.434(36.9)	1.633(48.8)	1.011(59.5)	1.279(53.9)

*The GA found result 0 but the priority rules didn't. Instead of showing the improvement, the best found by the priority rules is shown.

percentage improvement over the priority rules. Any of our results which are worse than the priority rules are shaded light gray. The GA results are worse than the priority rules in only 5 of 260 scenarios. In general, the relative improvement of the GA is larger in tight due date situations for weighted tardiness, maximum tardiness, weighted lateness, and weighted number of tardy jobs. For loose due date problems, although the priority rules yield similar results to the GA for the objective functions which only involve tardiness, the GA outperforms the priority rules for all objective functions giving a credit or penalty to earliness -- *e.g.*, for weighted lateness and for weighted earliness plus weighted tardiness. For the nonregular objective, *i.e.*, weighted earliness plus weighted tardiness, the priority rules perform much worse than the GA because such a nonregular objective is harder for priority rules to optimize, given that they consider only a few pieces of local information about jobs or machines, without the awareness of global information. This result also shows the superiority of the GA for the nonregular objective function.

6.3 Summary

This chapter extends the GA-based scheduling system for static JSSPs to dynamic JSSPs in which jobs arrive continually. The idea of a decomposition approach with rescheduling using a modification of the adapted population is quite general, and can be implemented for dynamic JSSPs with other stochastic events such as machine breakdowns, job cancellations, due date changes, etc. For example, when a job is cancelled, the information about the cancelled job can be removed to create a smaller problem. The individuals of the old population are then modified by applying the G&T algorithm to the newly generated problem, with the unaltered elements following their

relationships in the old individuals. Thus, the relationships among the operations are preserved to the extent possible. The experimental results show that a significant improvement over priority rule approaches was achieved for both deterministic and stochastic dynamic JSSPs using a genetic algorithm approach, and at a reasonable additional computational cost. Consider the results for various objective functions: while no one priority rule dominated other priority rules for all objective functions, our approach consistently outperformed the priority rules. Such a consistent superiority shows the robustness of the GA to the objective functions. Another interesting result concerns the manufacturing environment. Raman *et al.* [57] reported that the selection of a different and appropriate scheduling rule improves the system performance under different manufacturing environments. In contrast to that claim, our approach outperformed the priority rules with respect to the machine workload, imbalance of machine workload, and due date tightness. This shows the robustness of the GA to the manufacturing environment on the JSSPs. The aspect of the robustness of the GA to the objective functions and the manufacturing environment is more prominent and more important in real-world manufacturing systems.

CHAPTER 7

Conclusions

The JSSP is one of the most general and most difficult of all traditional scheduling problems. It has led to many techniques emanating from the fields of artificial intelligence and operations research which provide approximate or exact solutions to JSSPs. Throughout this dissertation, we have developed a GA-based scheduling system for JSSPs, especially dynamic JSSPs. Although the problems studied are simplified models of real-world scheduling problems, the impressive results show the promise for the scheduling system as the basis of an approach to real-world scheduling problems.

7.1 Summary

The primary objective of this research is to develop an efficient GA-based scheduling system to address JSSPs, especially dynamic JSSPs. To achieve this goal, first, the representation scheme and the genetic operators were designed to address the validity problem. In the scheduling system, the representation scheme was a direct representation, which encodes the operation starting times. The two G&T-algorithm-based genetic operators, THX crossover and mutation, were designed to better transmit the temporal relationships in the schedule. The designed scheduling framework was tested on some standard benchmark JSSPs. The superior results indicate the successful incorporation of

problem-specific knowledge into the genetic operators, and show the effectiveness of the scheduling system.

We further investigated PGAs for JSSPs by comparing single-population GAs and PGAs on the FT10x10 problem and reported on various PGA models. In general, the effect of parallelizing GAs is twofold. First, PGAs can alleviate the premature convergence problem and improve the results. Second, PGAs can often find the same quality of results in a fewer number of evaluations compared to single-population GAs. Furthermore, in cgGAs, the number of islands has a greater positive effect on performance than simply increasing population size. Additionally, a good connection topology can further increase the performance. Best results were obtained with the hybrid PGA consisting of cgGAs connected in a fgGA-style topology.

The scheduling system was then extended to address dynamic JSSPs. In deterministic dynamic JSSPs, the scheduling system was tested on a set of benchmark problems with respect to different objective functions. Two versions of GAs were applied. One was the single-population GA; the other was the hybrid PGA which achieved best performance in static JSSPs. The results were compared with priority rules and another GA-based scheduling system. The hybrid PGA was seen to provide the best results across a variety of schedule quality criteria.

In stochastic dynamic JSSPs, we decomposed the stochastic JSSP into a series of deterministic problem. A deterministic problem is generated whenever a new job enters the system. At each such point in time, the job information is updated. We have also presented an innovative rescheduling method which modifies the adapted population into a new population between successive events. The temporal relationships among the

operations in each individual of the adapted population are preserved. To evaluate the scheduling system, a simulation model of stochastic JSSPs representing a variety of various manufacturing environments was designed. Two versions of GA were compared with priority rules for the objective of minimizing weighted flow time. One used rescheduling from scratch; the other used the newly designed rescheduling method. Both versions of the GA outperformed the priority rules, and the GA with the newly designed rescheduling method performed better than the GA with rescheduling from scratch. The results indicate that the rescheduling method successfully preserves the information of the temporal relationships and enhances the efficiency of genetic search. In other objective functions, we compared the scheduling system with priority rules. The experimental results show that the scheduling system outperformed the priority rules for all objective functions with respect to the machine workload, imbalance of machine workload, and due date tightness.

7.2 Contributions

The main contributions of this dissertation are summarized as follows:

(1) Given the computing power available, it becomes increasingly possible and more important to design effective approaches to obtain better schedules, even at additional computational cost. This research has presented one such approach. The GA-based scheduling system is successful on a wide variety of benchmark problems and simulation problems. Compared to traditional priority rules, the scheduling system yields significant improvement, at a reasonable additional computational cost.

(2) In contrast to previous GA approaches, the design of the genetic operators and the rescheduling method developed here focuses on the schedule level instead of the chromosome level. The excellent results indicate that such a design can effectively incorporate the problem-specific knowledge into the GA and enhance genetic search.

(3) The effect of parallelizing GAs has been investigated. For these problems, PGAs not only alleviate the premature convergence problem and improve the results, but also can find the same quality of results in a fewer number of evaluations compared to single-population GAs. The proposed hybrid PGA which consists of cgGAs connected in a fgGA-style topology performs best, appearing to integrate successfully the advantages of cgGAs and fgGAs.

(4) The robustness of the GA-based scheduling system over various schedule quality criteria has been demonstrated. In this research, no one priority rule dominated other rules for all objective functions. The GA-based scheduling system consistently outperformed the priority rules across all objective functions. The results indicate that the GA-based scheduling system is robust with regard to the objective function and is a powerful general job shop scheduling tool.

(5) The robustness of the GA-based scheduling system over various manufacturing environments has been demonstrated. In this research, the GA-based scheduling system outperformed the priority rules with respect to the machine workload, imbalance of machine workload, and due date tightness. The results show that the GA-based scheduling system is robust with regard to the manufacturing environment.

(6) The decomposition approach with rescheduling from a modification of the adapted population is quite general, and can be implemented for dynamic JSSPs with other nondeterministic events such as machine breakdowns, job cancellations, due date changes, etc. It is easy to see how the scheduling system can be easily extended to deal with these events. Therefore, the scheduling system can be a promising basis for an approach to real-world scheduling problems.

7.3 Future Research

With the framework presented by this research, many interesting directions for future research exist.

(1) There are some possible ways to further improve the performance of the GA-based scheduling system. In recent GA research, the incorporation of local search (Section 2.3) into GAs has shown promise for enhancing genetic search [80, 90, 38, 110]. The incorporation can be done in the evaluation stage. After a schedule is decoded from the chromosome, some local search technique would be applied to the schedule. The re-optimized schedule would then be encoded back into the chromosome. Thus, the search space is restricted to local optima. Whether a GA benefits from local search highly depends on the problem under consideration. Further investigation is necessary. Another possible way to improve the performance of the scheduling system is to apply a more sophisticated migration strategy to the hybrid PGA. The migration strategy can be based on the diversity measure of each subpopulation or similarity between subpopulations. The main purpose is to maintain the diversity of the total population.

(2) In the simulation model of stochastic dynamic JSSPs, because the load of the job shop is zero at the beginning and the end of the run, the expected load is not maintained during these two periods. We can investigate the influence of different loads more accurately by preloading jobs in the shop and observing only some interval in the run.

(3) The rescheduling is done at the time of arrival of each job. Another possible way is to reschedule jobs only at specified intervals and to allow new jobs to wait in the shop. The penalty caused by queueing new jobs should depend on the length of the interval and the characteristics of shop load. It is worthwhile to compare these two rescheduling approaches.

(4) Although the scheduling system can be easily extended to deal with other nondeterministic events, real world scheduling problems are more complex. By further relaxing assumption (A5), (A8), and (A14) in Section 2.2, sequence-dependent setup time, parallel machines, and alternative plans are allowed. It is necessary to develop efficient methods to deal with these extensions. One possible method is to extend the representation to incorporate the selection of alternative plans and parallel machines. Another possible method is to introduce some heuristics into the evaluation stage to choose the process plan as well as the parallel machine.

(5) To make the scheduling system easy to use, a good user interface is important. Besides facilitating the input of job information and the output of the generated schedules by the GA, this user interface module may provide the user with a number of algorithms or heuristics for comparison. After showing the schedule, the module should allow the user to manually edit the schedule and ask the

scheduling system to re-optimize the modified schedule. Such modification can be treated as a nondeterministic event and is easy to process in the scheduling system.

(6) A more thorough evaluation of the scheduling system on some real-world scheduling problems is important. This is essential prior to adopting the scheduling system in practice.

BIBLIOGRAPHY

BIBLIOGRAPHY

[1] Garey, M.R. and Johnson, D.S., *Computers and Intractability: A Guide to the Theory of NP-Completeness*, Freeman, San Francisco, CA, 1979.

[2] Mellor, P., "A Review of Job Shop Scheduling," *Operational Research Quarterly*, 17, 2, pp. 161-171, 1966.

[3] Lawler, E.L., Lenstra, J.K., and Rinnooy Kan, A.H.G., "Recent Development in Deterministic Sequencing and Scheduling: A Survey," *Deterministic and Stochastic Scheduling*, Dempster, M. A. H. et al. (eds), pp. 35-73, Reidel, Dordrecht, 1982.

[4] Coffman, E.G., *Computer and Job-Shop Scheduling Theory*, John Willey & Sons, New York, NY, 1976.

[5] French, S., *Sequencing and Scheduling: An Introduction to the Mathematics of the Job-Shop*, John Willey & Sons, New York, NY, 1982.

[6] Baker, K.R., *Introduction to Sequencing and Scheduling*, John Willey & Sons, New York, NY, 1974.

[7] Conway, R.W., Maxwell, W.L., and Miller, L.W., *Theory of Scheduling*, Addison-Wesley, Reading, MA, 1967.

[8] Muth, J.F. and Thompson, G.L., *Industrial Scheduling*, Prentice-Hall, Englewood Cliffs, NJ, 1963.

[9] Holland, H.J., *Adaptation in Natural and Artificial Systems*, MIT Press, Cambridge, MA, 1992.

[10] Pinedo, M., *Scheduling: Theory, Algorithms, and Systems*, Prentice-Hall, Englewood Cliffs, NJ, 1975.

[11] Nononha, S.J. and Sarma, V.V.S., "Knowledge-Based Approaches for Scheduling Problems: A Survey," *IEEE Transactions on Knowledge and Data Engineering*, Vol. 3, No. 2, pp. 160-171, 1991.

[12] Roadmmer, F.A. and White, K.P., "A Recent Survey of Production Scheduling," *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. 18, No. 6, pp. 841-851, 1988.

[13] Graves, S.C., "A Review of Production Scheduling," *Operations Research*, Vol. 29, No. 4, pp. 646-676, 1981.

[14] Righter, R., "Scheduling," *Stochastic Orders*, Shaked, M. and Shanthikumar, G. (eds), pp. 381-432, Academic Press, San Diego, CA, 1994.

[15] Herrmann, J.W. and Lee, C., "A Classification of Static Scheduling Problems," *Complexity in Numerical Optimization*, Pardalos, P.M. (ed), pp. 203-253, World Scientific, Singapore, 1993.

[16] Panwalkar, S.S. and Iskander, W., "A Survey of Scheduling Rules," *Operations Research*, Vol. 25, No. 1, pp. 45-61, 1977.

[17] Bhaskaran, K. and Pinedo, M, "Dispatching," *Handbook of Industrial Engineering*, Salvendy, G. (ed.), pp. 2184-2198, Wiley, New York, NY, 1992.

[18] Glover, F., "Tabu Search: A Tutorial," *Interfaces*, Vol. 20, No. 4, pp. 74-94, 1990.

[19] Kirkpatrick, S., Gelatt Jr., C.D., and Vecchi, M.P., "Optimization by Simulated Annealing," *Science*, Vol. 220, pp. 671-680, 1983.

[20] Dell'Amico, M. and Trubian, M., "Applying Tabu-Search to the Job Shop Scheduling Problem," *Annals of Operations Research*, Vol. 41, pp. 231-252, 1991.

[21] Foo, Y.S. and Takefuji, Y., "Integer Linear Programming Neural Networks for Job-Shop Schedule," *IEEE ICNN*, pp. 341-348, 1989.

[22] Zhou, D.N., Cherkassky, V., Baldwin, T.R., and Hong, D.W., "Scaling Neural Network for Job-Shop Scheduling," *IEEE IJCNN*, pp. 889-894, 1990.

[23] Lo, Z. and Bavarian, B., "Multiple Job Scheduling with Artificial Neural Networks," *Computers Elect. Eng.*, Vol. 19, No. 2, pp. 87-101, 1993.

[24] Tank, D. and Hopfield, J., "Simple Neural Optimization Networks: An A/D Converter, Signal Decision Circuit, and a Linear Programming Circuit", *IEEE Trans. on Circuits and Systems*, Vol. CAS-33, No. 5, pp. 533-541, 1986.

[25] Wallace, M., "Applying Constraints for Scheduling," *Constraint Programming*, Mayoh, B., Tyugu, E. and Penjaam, J. (eds), NATO Advanced Science Institute Series, Springer-Verlag, 1994.

[26] Fox, M.S. and Smith, S.F., "ISIS: A Knowledge-Based System for Factory Scheduling," *Expert Systems*, Vol. 1, No. 1, pp. 25-49, 1984.

[27] Van Hentenryck, P., *Constraint Satisfaction in Logic Programming*, MIT Press, Cambridge, MA, 1989.

[28] Smith, S.F., Fox, M.S., and Ow, P.S., "Constructing and Maintaining Detailed

Production Plans: Investigations into the Development of Knowledge-Based Factory Scheduling Systems," *AI Magazine*, Vol. 7, No. 4, pp. 45-61, 1986.

[29] Le Pape, C., "Implementation of Resource Constraints in ILOG SCHEDULE: A Library for the Development of Constraint-Based Scheduling Systems," *Intelligent Systems Engineering*, Vol. 3, No. 2, pp. 55-66, 1994.

[30] Goldberg, D., *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison-Wesley, Reading, MA, 1989.

[31] Davis, L., "Job-Shop Scheduling with Genetic Algorithms," *Proc. Int'l Conf. on Genetic Algorithms and their Applications*, pp. 136-149, Lawrence Erlbaum, Hillsdale, NJ, 1985.

[32] Cleveland, G.A. and Smith, S.F., "Using Genetic Algorithms to Schedule Flow Shop Releases," *Proc. Third Int'l Conf. on Genetic Algorithms*, pp. 160-169, Morgan Kaufmann, San Mateo, CA, 1989.

[33] Whitley, D., Starkweather, T, and Shaner, D., "The Traveling Salesman and Sequence Scheduling: Quality Solutions Using Genetic Edge Recombination," *Handbook of Genetic Algorithms*, Davis, L. (ed), pp. 350-372, Van Nostrand Reinhold, New York, NY, 1991.

[34] Bagchi, S., Uckun, S., Miyabe, Y., and Kawamura, K., "Exploring Problem-Specific Recombination Operators for Job Shop Scheduling," *Proc. Fourth Int'l Conf. on Genetic Algorithms*, pp. 10-17, Morgan Kaufmann, San Mateo, CA, 1991.

[35] Nakano, R. and Yamada, T. "Conventional Genetic Algorithms for Job-Shop Problems," *Proc. Fourth Int'l Conf. on Genetic Algorithms*, pp. 474-479, Morgan Kaufmann, San Mateo, CA, 1991.

[36] Yamada, T. and Nakano, R. "A Genetic Algorithm Applicable to Large-Scale Job-Shop Problems," *Parallel Problem Solving from Nature, 2*, pp. 281-290, North-Holland, Amsterdam, 1992.

[37] Tamaki, H. and Nishikawa, Y., "A Parallel Genetic Algorithm based on a Neighborhood Model and It's Application to the Jobshop Scheduling," *Parallel Problem Solving from Nature, 2*, pp. 573-582, North-Holland, Amsterdam, 1992.

[38] Uckun, S., Bagchi, S., and Kawamura, K., "Managing Genetic Search in Job Shop Scheduling," *IEEE Expert*, Vol. 8, No. 5, pp. 15-24, 1993.

[39] Paredis, J., "Exploiting Constraints as Background Knowledge for Genetic Algorithms: a Case-study for Scheduling," *Parallel Problem Solving from Nature, 2*, pp. 229-238, North-Holland, Amsterdam, 1992.

[40] Fang, H., Ross, P. and Corne, D., "A Promising Genetic Algorithm Approach to Job-Shop Scheduling, Rescheduling, and Open-Shop Scheduling Problems," *Proc.*

Fifth Int'l Conf. on Genetic Algorithms, pp. 375-382, Morgan Kaufmann, San Mateo, CA, 1993.

[41] Syswerda, G., "Schedule Optimization Using Genetic Algorithms," *Handbook of Genetic Algorithms*, Davis, L. (ed), pp. 332-349, Van Nostrand Reinhold, New York, NY, 1991.

[42] Bruns, R., "Direct Chromosome Representation and Advance Genetic Operators for Production Scheduling," *Proc. Fifth Int'l Conf. on Genetic Algorithms*, pp. 352-359, Morgan Kaufmann, San Mateo, CA, 1993.

[43] Goldberg, D.E. and Lingle Jr., R., "Alleles, Loci, and the Traveling Salesman Problem", *Proc. First Int'l Conf. on Genetic Algorithms and their Applications*, pp. 154-159, Lawrence Erlbaum, Hillsdale, NJ, 1985.

[44] Grefenstette, J., Gopal, R., Rosmaita, B., and Van Gucht, D., "Genetic Algorithms for the Traveling Salesman Problem," *Proc. First Int'l Conf. on Genetic Algorithms and their Applications*, pp. 160-165, Lawrence Erlbaum, Hillsdale, NJ, 1985.

[45] Grefenstette, J., "Incorporating Problem Specific Knowledge in Genetic Algorithms," *Genetic Algorithms and Simulated Annealing*, Davis, L. (ed), pp. 42-60, Morgan Kaufmann, Los Altos, CA, 1987.

[46] Giffler, J. and Thompson, G.L., "Algorithms for Solving Production Scheduling Problems," *Operations Research*, Vol. 8, pp. 487-503, 1960.

[47] Lin, S.-C., Punch, W.F., and Goodman, E.D., "Coarse-Grain Parallel Genetic Algorithms: Categorization and New Approach," *IEEE SPDP*, pp. 28-39, 1994.

[48] Redder, D., Punch, W.F., Lin, S.-C., Tomanek, D., and Kim, S-G, "Using Genetic Algorithms to Find Low Energy Configurations of Large Molecules", GARAGe Technical Report GARAGe94-3 (MSU Master's Project Report), 1994.

[49] Lin, S.-C., Punch, W.F., and Goodman, E.D., "A Hybrid Model Utilizing Genetic Algorithms and Hopfield Neural Networks for Function Optimization", GARAGe Technical Report GARAGe95-01-02, Michigan State Universtity, 1995.

[50] Punch, W.F., Averill, R.C., Goodman, E.D., Lin, S.-C., and Ding, Y., "Design Using Genetic Algorithms -- Some Results for Laminated Composite Structures", accepted for publication in *IEEE Expert*, Jan, 1995.

[51] Butler, R., and Lusk, E., "User's Guide to the p4 Programming System" Technical Report ANL-92/17, Argonne National Laboratory, Argonne, IL, 1992.

[52] Schraudolph, N, and Grefenstette, J., "A User's Guide to GAucsd 1.4", Technical Report CS92-249, CSE Department, UC San Diego, La Jolla, CA, 1992.

[53] Morton, T.E., and Pentico, D.W., *Heuristic Scheduling Systems*, John Wiley &

Sons, 1993.

[54] Schroeder, G.R., *Operations Management: Decision Making in the Operations Function*, 4th ed., McGraw-Hill, New York, NY, 1993.

[55] Dervitsiotis, N.K., Operations Management, McGraw-Hill, New York, NY, 1981.

[56] Raman, N., Rachamadugu, R.V., and Talbot, F.B., "Real-time Scheduling of an Automated Manufacturing center," *European Journal of Operational Research*, vol. 40, pp. 222-242, 1989.

[57] Raman, N., Talbot, F.B., Rachamadugu, R.V., "Due Date Based Scheduling in a General Flexible Manufacturing System," *Journal of Operations Management*, vol. 8, no. 2, pp. 115-132, 1989.

[58] Raman, N., and Talbot, F.B., "The Job Shop Tardiness Problem: a Decomposition Approach," *European Journal of Operational Research*, vol. 69, pp. 187-199, 1993.

[59] Fang, H., "Genetic Algorithms in Timetabling and Scheduling," Ph.D. thesis, Department of Artificial Intelligence, University of Edinburgh, 1994.

[60] Garey, M.R., Johnson, D.S., and Sethi, R., "The Complexity of Flowshop and Jobshop scheduling," *Math. Oper. Res.*, vol. 1, pp. 117-129, 1976.

[61] Bäck, T., *Evolutionary Algorithms in Theory and Practice: Evolutionary Strategies, Evolutionary Programming, Genetic Algorithms*, Oxford University Press, 1996.

[62] Grefenstette, J.J. and Baker, J.E., "How Genetic Algorithms Work: A Critical Look at Implicit Parallelism," *Proc. Third Int'l Conf. on Genetic Algorithms*, pp. 20-27, Morgan Kaufmann, San Mateo, CA, 1989.

[63] Goodman, E. D. *An Introduction to GALOPPS -- the Genetic ALgorithm Optimized for Portability and Parallelism System, Release 3.2,* Technical Report http://isl.cps.msu.edu/GA/papers/GARAGe96-07-01.ps, Genetic Algorithms Research and Applications Group, Michigan State University, 1996.

[64] Forrest, S., "Documentation for Prisoner's Dilemma and Norms Programs that use the Genetic Algorithm," Technical Report, Univ. of Michigan, 1985.

[65] Baker, J.E., "Reducing Bias and Inefficiency in the Selection Algorithms," *Genetic Algorithms and Their Applications: Proc. Second Int'l Conf. on Genetic Algorithms*, pp. 14-21, Lawrence Erlbaum, Hillsdale, NJ, 1987.

[66] Baker, J.E., "Adaptive Selection Methods for Genetic Algorithms," *Proc. Int'l Conf. on Genetic Algorithms and Their Applications*, pp. 101-111, Lawrence Erlbaum, Hillsdale, NJ, 1985.

[67] Brindle, A., *Genetic Algorithms for Function Optimization*, Unpublished Ph.D.

thesis, University of Alberta, Edmonton, Canada, 1981.

[68] Syswerda, G., "Uniform Crossover in Genetic Algorithms," *Proc. Third Int'l Conf. on Genetic Algorithms*, pp. 2-9, Morgan Kaufmann, San Mateo, CA, 1989.

[69] Davis. L., "Oder-Based Genetic Algorithms and the Graph Coloring Problem," *Handbook of Genetic Algorithms*, Davis, L. (ed), pp. 72-90, Van Nostrand Reinhold, New York, NY, 1991.

[70] Goldberg, D.E. and Lingle Jr., R., "Alleles, Loci, and the Travelling Salesman Problem," *Proc. Int'l Conf. on Genetic Algorithms and Their Applications*, pp. 154-159, Lawrence Erlbaum, Hillsdale, NJ, 1985.

[71] Whitley, D., Starkweather, T., and Fuquay, D., "Scheduling Problems and Traveling Salesmen: The Genetic Edge Recombination Operator," *Proc. Third Int'l Conf. on Genetic Algorithms*, pp. 133-140, Morgan Kaufmann, San Mateo, CA, 1989.

[72] Oliver I.M., Smith, D.J., and Holland, J.R.C., "A Study of Permutation Crossover Operators on the Traveling Salesman Problem," *Genetic Algorithms and Their Applications: Proc. Second Int'l Conf. on Genetic Algorithms*, pp. 224-230, Lawrence Erlbaum, Hillsdale, NJ, 1987.

[73] Whitley, D., "GENITOR: a Different Genetic Algorithm," *Proc. Rocky Mountain Conf. on Artificial Intelligence*, pp. 118-130, Denver, CO, 1988.

[74] Bierwirth, C., Kropfer, H., Mattfeld, D.C., and Rixen, I., "Genetic Algorithm based Scheduling in a Dynamic Manufacturing Environment," *IEEE Conf. on Evolutionary Computation*, Perth, IEEE Press, 1995.

[75] Blackstone, J.H., Philips, D.T., and Hogg, G.L., "A State-of-art Survey of Dispatching Rules for Manufacturing Job Shop Operations," *International Journal of Production Research*, vol. 20, no. 1, pp. 27-45, 1982.

[76] Kiran, A.S., "Simulation Studies in Job Shop Scheduling - I: A Survey," Computers & Industrial Engineering, vol. 8, pp. 87-93, 1984.

[77] Kiran, A.S., "Simulation Studies in Job Shop Scheduling - II: Performance of Priority Rules," *Computers & Industrial Engineering*, vol. 8, pp. 95-105, 1984.

[78] Vaessens, R.J.M., Aarts, E.H.L., and Lenstra, J.K., "A Local Search Template," *Parallel Problem Solving from Nature, 2*, pp. 65-74, North-Holland, Amsterdam, 1992.

[79] Vaessens, R.J.M., Aarts, E.H.L., and Lenstra, J.K., "Job Shop Scheduling by Local Search," Memorandum COSOR 94-05, 2nd revised version, Eindhoven University of Technology, Netherlands.

[80] Dorndorf, U. and Pesch, E. "Combining Genetic- and Local Search for Solving the

Job Shop Scheduling Problem," *APMOD93 Proc. Preprints*, pp. 142-149, Budapest, Hungary, 1993.

[81] Dorndorf, U. and Pesch, E. "Evolution Based Learning in a Job Shop Scheduling Environment," *Computers & Operations Research*, vol. 22, no. 1, pp. 25-40, 1995.

[82] Kobayashi, S., Ono, I., and Yamamura, M. "An Efficient Genetic Algorithm for Job Shop Scheduling Problems," *Proc. Sixth Int'l Conf. on Genetic Algorithms*, pp. 506-511, Morgan Kaufmann, San Mateo, CA, 1995.

[83] Storer, R. H., Wu, S.D., and Vaccari, R. "New Search Spaces for Sequencing Problems with Application to Job Shop Scheduling," *Management Science,* vol. 38, pp. 1495-1509, 1992.

[84] Yamada, T. and Nakano, R. "A Genetic Algorithm Applicable to Large-Scale Job-Shop Problems," *Parallel Problem Solving from Nature, 2*, pp. 281-290, North-Holland, Amsterdam, 1992.

[85] Park, L. and Park, C.H., "Genetic Algorithm for Job Shop Scheduling Problems Based on Two Representational Schemes," Electronics Letters, vol. 31, no. 23, pp. 2051-2053, 1995.

[86] Davidor, Y., Yamada, T., and Nakano, R., "The ECOlogical Framework II: Improving GA Performance at Virtually zero cost," *Proc. Fifth Int'l Conf. on Genetic Algorithms*, pp. 171-176, Morgan Kaufmann, San Mateo, CA, 1993.

[87] Croce, F.D., Tadei, R., and Volta, G., "A Genetic Algorithm for the Job Shop Problem," *Computers & Operations Research*, vol. 22, no. 1, pp. 15-24, 1995.

[88] Falkenauer, E. and Bouffouix, S., "A Genetic Algorithm for Job-Shop," *Proc. 1991 IEEE Int'l Conf. on Robotics and Automation*, Sacramento, CA, 1991.

[89] Kim, G.H. and Lee, C.S.G., "An Evolutionary Approach to the Job-shop Scheduling Problem," *Proc. 1994 IEEE Int'l Conf. on Robotics and Automation*, vol 1, pp. 501-506, San Diego, CA, 1994.

[90] Mattfeld, D.C., Kopfer, H., and Bierwirth, C. "Control of Parallel Population Dynamics by Social-Like Behavior of GA-Individuals," *Parallel Problem Solving from Nature, 3*, pp. 15-24, Springer-Verlag, Berlin, Heidelberg, 1994.

[91] Bierwirth, C. "A Generalized Permutation Approach to Job Shop Scheduling with Genetic Algorithms," OR-Spektrum, Special Issue: Applied Local Search, Pesch, E. and Vo, S. (eds), vol. 17, No. 213, pp. 87-92, 1995.

[92] Bierwirth, C., Kropfer, H., Mattfeld, D.C., and Rixen, I., "Genetic Algorithm based Scheduling in a Dynamic Manufacturing Environment," *IEEE Conf. on Evolutionary Computation*, Perth, IEEE Press, 1995.

[93] Rixen, I., Bierwirth, C., and Kopfer, H., "A Case Study of Operational Just-in-Time Scheduling using Genetic Algorithms," *Evolutionary Algorithms in Management Application*, Biethahn, J. and Nissen, V. (eds), pp. 112-123, Springer-Verlag, 1995.

[94] Adams, J., Balas, E., and Zawack, D. "The Shifting Bottleneck Procedure in Job Shop Scheduling," *Management Science*, vol. 34, pp. 391-401, 1988.

[95] Grabowski, J., Nowicki, E., and Zdrzalka, S. "A Block Approach for Single Machine Scheduling with Release Date and Due Date," *European J. Oper. Res.*, vol. 26, pp. 278-285, 1986.

[96] Bierwirth, C., Kropfer, H., Mattfeld, D.C., and Rixen, I., "Genetic Algorithm based Scheduling in a Dynamic Manufacturing Environment," *IEEE Conf. on Evolutionary Computation*, Perth, IEEE Press, 1995.

[97] Goldberg, D. and Segrest, P., "Finite Markov Chain Analysis of Genetic Algorithms," *Genetic Algorithms and Their Applications: Proc. Second Int'l Conf. on Genetic Algorithms*, pp. 1-8, Lawrence Erlbaum, Hillsdale, NJ, 1987.

[98] De Jong, K., *An Analysis of the Behavior of a Class of Genetic Adaptive Systems*, University of Michigan, 1975.

[99] Mauldin, M., "Maintaining Diversity in Genetic Search," *Nat. Conf. on Artificial Intelligence*, pp. 247-250, 1984.

[100] Goldberg, D.E. and Richardson, T., "Genetic Algorithms with Sharing for Multimodal Function Optimization," *Genetic Algorithms and Their Applications: Proc. Second Int'l Conf. on Genetic Algorithms*, pp. 41-49, Lawrence Erlbaum, Hillsdale, NJ, 1987.

[101] Crow, J.F., *Basic Concepts in Population, Quantitative, and Evolutionary Genetic*, W.H. Freeman and Company, New York, 1986.

[102] Pettey, C., Leuze, M., and Grefenstette, J., "A Parallel Genetic Algorithm," *Genetic Algorithms and Their Applications: Proc. Second Int'l Conf. on Genetic Algorithms*, pp. 155-161, Lawrence Erlbaum, Hillsdale, NJ, 1987.

[103] Starkweather, T., Whitley, D., and Mathias, K., "Optimization Using Distributed Genetic Algorithms," *Parallel Problem Solving from Nature*, pp. 176-185, Springer-Verlag, Berlin, Heidelberg, 1990.

[104] Tanese, R., "Distributed Genetic Algorithms," *Proc. Third Int'l Conf. on Genetic Algorithms*, pp. 434-440, Morgan Kaufmann, San Mateo, CA, 1989.

[105] Manderick, B. and Spiessens, P. "Fine-Grained Parallel Genetic Algorithms," *Proc. Third Int'l Conf. on Genetic Algorithms*, pp. 428-433, Morgan Kaufmann, San Mateo, CA, 1989.

[106] Muhlenbein, H. "Parallel Genetic Algorithms, Population Genetics and Combinatorial Optimization," *Proc. Third Int'l Conf. on Genetic Algorithms*, pp. 416-421, Morgan Kaufmann, San Mateo, CA, 1989.

[107] Juels, A. and Wattenberg, M. "Stochastic Hillclimbing as a Baseline Method for Evaluating Genetic Algorithms," Technical Report csd-94-834, University of California at Berkeley, 1994.

[108] Lin, S.-C., Goodman, E.D., and Punch, W.F., "Investigating Parallel Genetic Algorithms on Job Shop Scheduling Problems," Technical Report, Genetic Algorithms Research and Applications Group, Michigan State University, 1996.

[109] Applegate, D. and Cook, W., "A Computational Study of the Job-Shop Scheduling Problem," *ORSA Journal on Computing*, vol. 3, no. 2, pp. 149-156, 1991.

[110] Axkley, D.H., *A Connectionist Machine for Genetic Hillclimbing*, Kluwer Academic, 1987.

[111] Dechter, R., "Constraint Networks: A Survey," Technical Report, Department of Information and Computer Science, University of California at Irvine, CA, 1991.

[112] Smith, S.F., Fox, M.S., and Ow, P.S., "Constructing and Maintaining Detailed Production Plans: Investigations into the Development of Knowledge-Based Factory Scheduling System," *AI Magazine*, vol. 7, no. 4, Fall 1986, pp. 45-61.

[113] Wang, G., Goodman, E. and Punch, W. F., "Simultaneous Multi-Level Evolution," GARAGe Technical Report GARAGe96-03-01, Michigan State University, Submitted to Evolutionary Computation, 1996.

[114] Goodman, E.D., "An Introduction to GALOPPS -- The "Genetic Algorithm Optimized for Portability and Parallelism" System," Releases 2.35, Nov. 1994, Technical Report GARAGe94-11-01, Michigan State University.

[115] Carlier, J. and Pinson, E., "An Algorithm for Solving the Job-shop Problem," *Management Science*, vol. 35, pp. 164-176, 1989.

[116] Baker, J.R. and McMahon, G.B., "Scheduling the General Job-shop," *Management Science*, vol. 31, pp. 594-598, 1985.

[117] Taillard, E., "Parallel Taboo Search Technique for the Jobshop Scheduling Problem," *ORSA Journal on Computing*, vol 6, pp. 108-117, 1993.

[118] Aarts, E.H.L., Van Laarhoven, P.J.M., Lenstra, J.K., and Ulder, N.L.J., "A Computational Study of Local Search Algorithms for Job Shop Scheduling," *ORSA Journal on Computing*, vol. 6, pp. 118-125, 1994.

[119] Yamada, T. and Nakano, R., "Job Shop Scheduling by Simulated Annealing Combined with Deterministic Local Search," Metaheuristics International

Conference, Hilton Breckenridge, Colorado USA, pp. 344-349, 1995.

[120] Wennink, M., Personal communication, (upper bounds found by a taboo search algorithm), 1995.

[121] Balas, E. and Vazacopoulos, A., "Guided Local Search with Shifting Bottleneck for Job Shop Scheduling," Management Science Research Report #MSRR-609, Graduate School of Industrial Administration, Carnegie-Mellon University, Pittsburgh, Pennsylvania, 1994.

[122] Schraudolph, N. and Grefenstette, J., *A User's Guide to GAucsd 1.4*, July, 1992.

[123] Schraudolph, N. and Belew, R., "Dynamic Parameter Encoding for Genetic Algorithms," *Machine Learning*, pp. 9-21, June, 1992.

[124] Shaefer, C.G., "The ARGOT Strategy: Adaptive Representation Genetic Optimizer Technique," *Genetic Algorithms and Their Applications: Proc. Second Int'l Conf. on Genetic Algorithms*, pp. 50-55, Lawrence Erlbaum, Hillsdale, NJ, 1987.

[125] Collins, R.J. and Jefferson, D.R., "Selection in Massively Parallel Genetic Algorithms," *Proc. Fourth Int'l Conf. on Genetic Algorithms*, pp. 249-256, Morgan Kaufmann, San Mateo, CA, 1991.

[126] Spiessens, P. and Manderick, B., "A Massively Parallel Genetic Algorithm: Implementation and First Analysis," *Proc. Fourth Int'l Conf. on Genetic Algorithms*, pp. 279-286, Morgan Kaufmann, San Mateo, CA, 1991.

[127] Gorges-Schleuter, M., "ASPARAGOS An Asynchronous Parallel Genetic Optimization Strategy," *Proc. Third Int'l Conf. on Genetic Algorithms*, pp. 422-427, Morgan Kaufmann, San Mateo, CA, 1989.

[128] Goldberg, D.E., "Sizing Populations for Serial and Parallel Genetic Algorithms," *Proc. Third Int'l Conf. on Genetic Algorithms*, pp. 70-79, Morgan Kaufmann, San Mateo, CA, 1989.