A Genetic Algorithm Approach to Dynamic Job Shop Scheduling Problems

Shyh-Chang Lin **Erik D. Goodman** **William F. Punch, III**

Genetic Algorithms Research and Applications Group
230 Engineering Building
Michigan State University
East Lansing, MI 48824

Abstract

This paper describes a genetic algorithm approach to the dynamic job shop scheduling problem with jobs arriving continually. Both deterministic and stochastic models of the dynamic problem were investigated. 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. In the stochastic model, 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.

1 INTRODUCTION

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, etc. Due to their dynamic nature, real world scheduling problems are rather computationally complex and are known to be strongly NP-hard -- i.e., the time required to compute an optimal schedule increases exponentially with the size of the problem. From a practical vantage, a large number of priority rule approaches have been proposed 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*. [1], given the computing power available today, it becomes more important to design effective approaches to obtain better schedules, even at additional computational cost. This paper describes one such approach.

Since Davis proposed the first GA-based technique to address scheduling problems in 1985 [3], GAs have been used with increasing frequency in the context of job shop scheduling problems (JSSPs). Much of the GA research on this subject studied only static JSSPs, in which all jobs are ready to start at time zero, 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. In this study, we develop a GA approach that is modified from the original static JSSP version to address deterministic JSSPs. The stochastic JSSP may be decomposed into a series of deterministic problems by the method proposed by Raman *et al* [13]. A deterministic problem is generated at each occurrence of a nondeterministic event, and then solved by the GA. This paper considers two important issues. First, both deterministic and stochastic JSSPs with different objectives are examined. In stochastic JSSPs, we further investigate the influence of the manufacturing environment, such as different machine workloads, imbalance of machine workload, and different flow allowances. The results are compared with priority rules to assess the relative performance and the robustness of the proposed approach. Second, we propose an innovative rescheduling method which modifies the adapted population into a new population between successive events. The temporal relations among the operations in each individual of the

Table 1: Notation

J	Number of available jobs
M	Number of machines
P_j	Total processing time of job j
R_j	Remaining processing time of job j
p_{jm}	Processing time of job j on machine m
$\bar{p}_m$	Average operation processing time on machine m
a_m	Available time of machine m
r_j	Release time of job j
r_{jm}	The earliest time at which operation (j, m) can start
d_j	Due date of job j
w_j	Weight of job j
n_j	Number of remaining operations of job j
C_j	Completion time of job j
L_j	Lateness of job $j = C_j - d_j$
T_j	Tardiness of job $j = max(L_j, 0)$
E_j	Earliness of job $j = max(-L_j, 0)$
U_j	Unit penalty of job $j = \begin{cases} 1 & if\ C_j > d_j \\ 0 & otherwise \end{cases}$

adapted population are preserved. This method is compared with rescheduling from scratch and yields impressive results. The remainder of the paper is organized as follows. Section 2 defines the dynamic JSSP studied. Section 3 describes our approach to dealing with dynamic JSSPs and the rescheduling problem. Sections 4 and 5 report computational results for the deterministic and stochastic problems, respectively. Section 6 presents our conclusions.

The notation used in this paper is shown in Table 1.

2 DYNAMIC JOB SHOP SCHEDULING PROBLEM

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 *J*x*M,* where *J* is the number of jobs and *M* is the number of machines. The operation of job *j* on machine *m* is denoted by operation (*j*, *m*). 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 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. Therefore, we consider six other objective functions, described in Table 2. For reporting purposes, we use the normalized value of the objective, which is also defined in Table 2. Except for the objective of weighted flow time, 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.

3 A GA-BASED SCHEDULING SYSTEM

3.1 THE APPROACH FOR STATIC JSSPs

The proposed approach to dynamic JSSPs is based on our original GA-based scheduling system devised for static JSSPs [11]. Here we briefly describe the approach for static JSSPs. Each individual is a direct representation which encodes for each operation (in index order) its starting time in a feasible schedule. The number of fields on the chromosome is the number of operations. Such a direct representation doesn't suffer from the problem of false competition-- different representations of the same schedule competing against one another -- which is found in some indirect representation schemes [7]. 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 [4], it is difficult to encode the schedule back to the chromosome. This advantage is more important in our approach because the genetic operators we designed work on the schedule level and the modified schedules need to be encoded back to the chromosomes. The genetic operators are inspired by the Giffler and Thompson (G&T) algorithm [8]. The G&T algorithm is a systematic approach to generate *active* schedules, in which no operations can be completed earlier without delaying other operations. We give a brief outline of the G&T algorithm in Figure 1. Some previous approaches which are G&T-algorithm-based can be found

Table 2: Objective Functions

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	$\sum_j w_j(E_j + T_j)$	$\left(\sum_j w_j(E_j + T_j)\right) / \left(\sum_j w_j p_j\right)$

```
Step 1:
    Let $\boldsymbol{C}$ contain the first schedulable operation of each job;
    Let $r_{jm} = 0$, for all operations $(j, m)$ in $\boldsymbol{C}$.
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 1: The Giffler and Thompson Algorithm

in [4,5,10,16,17]. Basically, the G&T algorithm is used as an interpreter to decode any offspring into an active schedule. At each decision point in the G&T algorithm (step 4 in Figure 1), the operation with the earliest starting time reported in the parental schedule is chosen to be scheduled next. In contrast to previous approaches, which work on the chromosome level, the designed time horizon exchange (THX) crossover works on the schedule level. THX crossover randomly selects a crossover point just like a standard crossover, but instead of using the crossover point to exchange two chromosomes, THX crossover uses the crossover point as a scheduling decision point mark in the G&T algorithm to exchange information between two schedules. Before the decision point mark, the temporal relations among operations are inherited from one parent. In the remaining portion, the temporal relations are inherited from the other parent to the extent possible (i.e., while maintaining a valid schedule). The mutation operator randomly selects two operations in the block -- a sequence of successive operations on the critical path which are on the same machine with at least two operations. These two operations are reversed. After the mutated child is generated, we apply the G&T algorithm to interpret the child.

This scheduling system has been tested on some static job shop benchmarks and produced excellent results [11]. We further tested many models and scales of parallel GA in the context of static JSSPs. In our previous experiments, the hybrid model consisting of coarse-grain GAs connected in a fine-grain-GA-style topology performed best, appearing to integrate successfully the advantage of coarse-grain and fine-grain GAs.

3.2 THE APPROACH IN DYNAMIC JSSPs

The scheduling system described in the previous subsection can be applied to dynamic JSSPs after a few modifications. For deterministic JSSPs, we modify step 1 of the G&T algorithm to take account of the job release times as follows:

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

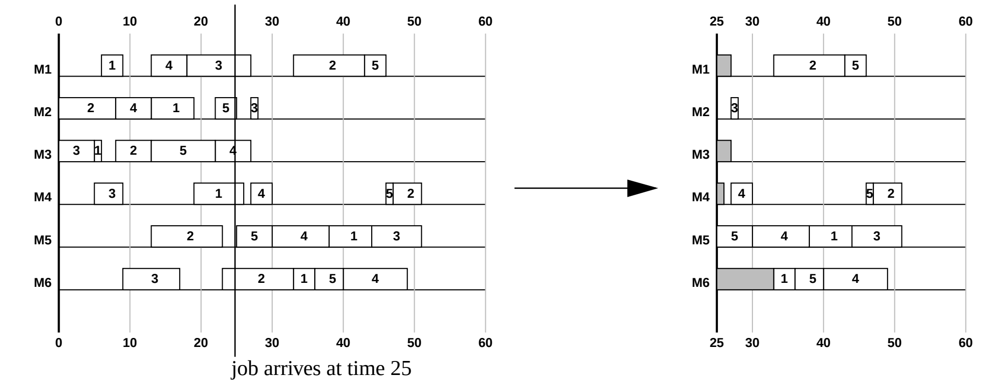
After the modification, the genetic operators are able to deal with deterministic JSSPs. To address stochastic JSSPs, we decompose the stochastic JSSP into a series of deterministic problems using the method proposed by Raman *et al* [13]. A deterministic problem is generated whenever a new job enters the system. At each such point in time, 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 processing 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. Assuming the machine is available at time a_m, we modify step 1 of the G&T algorithm to take into account the machine's 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 $\boldsymbol{C}$.
```

Figure 2 shows an example of the time decomposition method. A new job 6 arrives at the system at time 25. Machine 1, 3, 4, and 6 are blocked out because they are busy at time 25. Some operations are removed from the system and the information for job 6 is added to the new problem.

3.3 THE RESCHEDULING PROCESS

In stochastic JSSPs, after a new deterministic problem is generated at the time of a job arrival, it requires modifications in the existing schedule. Two methods can address the rescheduling problem. One is to discard the old population and construct the new schedule from scratch. This can be done simply by restarting the scheduling system with the new job shop problem. The other method uses a special feature of GAs which was observed by Bierwirth *et al.* [2]. Consider the final population in the last time period. 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 con-



Job	Release time	Operation 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

Job	Release time	Operation 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

Figure 2: A Time Decomposition Example

structing an individual in the initial population of the new problem, the temporal relations 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:

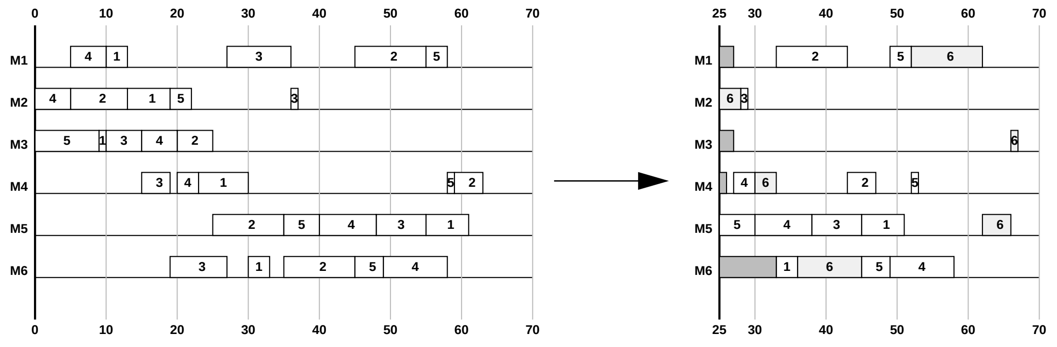
```
    Randomly select one operation from G.
    If the operation is from the new job(s), schedule it;
    else schedule the operation in G from the old problem with the earliest starting time reported in the
      individual of the adapted population.
```

Figure 3 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 2. The operations of the new job 6 are inserted among the other operations while the temporal relations among the old operations are preserved to the extent possible.

4 COMPUTATIONAL STUDY -- DETERMINISTIC PROBLEMS

4.1 EXPERIMENTAL DESIGN

Deterministic JSSPs can be solved in either an exact or heuristic manner. One example of an exact method is a depth-first branch-and-bound algorithm which builds a schedules forward in time [15]. 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



(a) A schedule in the last population

(b) A schedule in the initial population of the new problem

Figure 3: An Example of Modifying A Schedule From the Last Population To the Initial Population of the New Problem.

machines, such as processing times, due dates, release times, and machine loadings. Table 3 is a list of the priority rules used for comparison in this paper. The test set of 12 deterministic problems was taken from [12]. The objective functions examined are listed in Table 2. In minimizing the weighted flow time, we compared our GA approach with five priority rules -- RANDOM, FCFS, WSPT, WLWKR, and WTWORK rules. For the other objective functions, the priority rules used for comparison were WSPT, EGD, EOD, EMOD, MST, WS/OP, WCR, WCOVERT, and WR&M rules, except that in minimizing the maximum tardiness, 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 [6]. Fang's approach is based on a GA which uses a variant of an indirect representation devised for the traveling salesperson problem. 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.

The scheduling system described has been implemented in GALOPPS [9] and run in a Unix environment. 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 (SGA) with population size 50. The other was a parallel GA (PGA) in which 25 SGAs with subpopulation sizes of 20 were connected in a 5x5 torus. The migration interval was 50 generations. The number of generations of both versions was 50x(number of jobs).

4.2 RESULTS AND DISCUSSION

Results of the experiment are shown in Table 4. The reported results are the normalized values (see Table 2). The results shown in the "Pri." columns are the best results found by the priority rules, and the results reported under the remaining columns are the best results obtained from 10 runs of the corresponding GA for each problem. For the SGA and PGA, we also report the percentage improvement over the best results found by the priority rules and Fang. Our results which are worse than others are shaded light gray. Our SGA and PGA both performed consistently better than the priority rules. The SGA yields results better than or equal to Fang's in 61 of 72 scenarios. The PGA is seen to provide the best results. Only 3 of its 72 scenarios are worse than Fang's. The superior results show that the THX crossover and mutation successfully transmit useful characteristics -- *i.e.*, the temporal relationships among operations. Furthermore, the PGA performs better than the SGA because the premature convergence problem is alleviated by parallelizing the GA, allowing better global search.

5 COMPUTATIONAL STUDY -- STOCHASTIC PROBLEMS

5.1 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

Table 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	$1/[r_j + (d_j\text{-}r_j)R_j/P_j]$
EMOD	Earliest Modified Operational Due-date	$1/max(r_j+(d_j\text{-}r_j)R_j/Pj,\ t+p_{jm})$
MST	Minimum Slack Time	$-(d_j\text{-}R_j\text{-}t)$
WS/OP	Weighted Slack per OPeration	$w_j[1\text{-}(d_j\text{-}R_j\text{-}t)/n_j)]/p_{jm}$
WCR	Weighted Critical Ratio	$w_j[1\text{-}(d_j\text{-}t)/R_j)]/p_{jm}$
WCOVERT	Weighted COVERT	$w_j[1\text{-}(d_j\text{-}R_j\text{-}t)^+/2R_j)]^+/p_{jm}$
WR&M	Weighted R&M	$w_j exp[\text{-}(d_j\text{-}R_j\text{-}t)^+/2\overline{P}_m)]/p_{jm}$

Table 4: The normalized Results of the Deterministic Problems

Prob.	Size	Weighted Flow Time				Weighted Tardiness				Maximum Tardiness			
		Pri.	Fang	SGA	PGA	Pri.	Fang	SGA	PGA	Pri.	Fang	SGA	PGA
JB1	10x3	1.231	1.237	1.231(0.0)	1.231(0.0)	0.178	0.164	0.162(1.2)	0.162(1.2)	0.082	0.082	0.082(0.0)	0.082(0.0)
JB2	10x3	1.772	1.778	1.768(0.2)	1.768(0.2)	0.086	0.087	0.086(0.0)	0.086(0.0)	0.055	0.055	0.055(0.0)	0.055(0.0)
JB4	10x5	1.111	1.109	1.108(0.1)	1.108(0.1)	0.560	0.556	0.559(-0.5)	0.559(-0.5)	0.203	0.152	0.152(0.0)	0.152(0.0)
JB9	15x3	1.947	1.768	1.754(0.8)	1.754 (0.8)	0.185	0.177	0.169(4.5)	0.169(4.5)	0.067	0.061	0.044(27.9)	0.044(27.9)
JB11	15x5	1.795	1.794	1.723(4.0)	1.706 (4.9)	0.000	0.000	0.000(0.0)	0.000(0.0)	0.002	0.000	0.000(0.0)	0.000(0.0)
JB12	15x5	1.257	1.259	1.256(0.1)	1.256 (0.1)	0.218	0.139	0.139(0.0)	0.139(0.0)	0.071	0.060	0.060(0.0)	0.060(0.0)
LJB1	30x3	1.494	1.431	1.424(0.5)	1.391 (2.8)	0.276	0.215	0.224(-4.2)	0.190(11.6)	0.056	0.041	0.039(4.9)	0.032(22.0)
LJB2	30x3	1.924	1.826	1.783(2.4)	1.777 (2.7)	0.460	0.459	0.410(10.7)	0.395(13.9)	0.078	0.071	0.060(15.5)	0.060(15.5)
LJB7	50x5	1.692	1.669	1.623(2.8)	1.557 (6.7)	0.109	0.110	0.090(17.4)	0.060(45.0)	0.022	0.019	0.016(15.8)	0.016(15.8)
LJB9	50x5	2.490	2.659	2.475(0.6)	2.324 (6.7)	0.796	0.982	0.742(6.8)	0.615(22.7)	0.050	0.075	0.048(4.0)	0.039(22.0)
LJB10	50x8	1.776	1.728	1.731(-0.2)	1.697 (1.8)	0.479	0.455	0.478(-5.1)	0.438(3.7)	0.043	0.040	0.040(0.0)	0.034(15.0)
LJB12	50x8	2.207	2.138	2.115(1.1)	2.080 (2.7)	0.489	0.478	0.433(9.4)	0.399(16.5)	0.035	0.043	0.035(0.0)	0.031(11.4)

Prob.	Size	Weighted Lateness				Weighted Number of Tardy Jobs				Weighted Earliness plus Tardiness			
		Pri.	Fang	SGA	PGA	Pri.	Fang	SGA	PGA	Pri.	Fang	SGA	PGA
JB1	10x3	-0.173	-0.168	-0.173(0.0)	-0.173(0.0)	0.433	0.272	0.272(0.0)	0.272(0.0)	0.529	0.475	0.474(0.2)	0.474(0.2)
JB2	10x3	-0.812	-0.824	-0.816(-1.0)	-0.839(1.8)	0.097	0.097	0.097(0.0)	0.097(0.0)	0.901	0.758	0.690(9.0)	0.499(34.2)
JB4	10x5	0.497	0.490	0.493(-0.6)	0.493(-0.6)	0.709	0.688	0.688(0.0)	0.688(0.0)	0.622	0.620	0.621(-0.2)	0.621(-0.2)
JB9	15x3	-0.047	-0.073	-0.078(6.8)	-0.078(6.8)	0.425	0.238	0.236(0.8)	0.236(0.8)	0.395	0.384	0.369(3.9)	0.369(3.9)
JB11	15x5	-0.607	-0.655	-0.730(11.5)	-0.751(14.7)	0.000	0.078	0.000(0.0)	0.000(0.0)	0.415	0.263	0.262(0.4)	0.262(0.4)
JB12	15x5	-0.082	-0.102	-0.103(1.0)	-0.103(1.0)	0.431	0.434	0.431(0.0)	0.431(0.0)	0.494	0.247	0.246(0.4)	0.246(0.4)
LJB1	30x3	-0.112	-0.195	-0.206(5.6)	-0.214(9.7)	0.401	0.309	0.296(4.2)	0.296(4.2)	0.654	0.322	0.322(0.0)	0.279(13.4)
LJB2	30x3	0.013	-0.043	-0.077(79.1)	-0.078(81.4)	0.374	0.254	0.233(8.3)	0.233(8.3)	0.868	0.632	0.627(0.8)	0.601(4.9)
LJB7	50x5	-0.411	-0.414	-0.453(9.4)	-0.507(22.5)	0.294	0.180	0.167(7.2)	0.126(30.0)	0.515	0.374	0.345(7.8)	0.254(32.1)
LJB9	50x5	0.622	0.702	0.498(19.9)	0.354(43.1)	0.564	0.309	0.340(-10)	0.241(22.0)	0.935	1.178	0.833(10.9)	0.739(21.0)
LJB10	50x8	-0.038	-0.096	-0.082(-14.6)	-0.108(12.5)	0.533	0.399	0.419(-5.0)	0.399(0.0)	0.882	0.621	0.688(-10.8)	0.598(3.7)
LJB12	50x8	0.256	0.221	0.192(13.1)	0.113(48.9)	0.478	0.286	0.286(0.0)	0.274(4.2)	0.667	0.607	0.584(3.8)	0.461(24.1)

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 the due-date-related objective functions.

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 -- *i.e.*, 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 SGA were examined. The first reschedules the new deterministic problem from scratch. The second reschedules by using a modified population. For the other objective functions, we applied only the second SGA to compare with the priority rules.

5.2 RESULTS AND DISCUSSION

The results of the weighted flow time objective are shown in Table 5. We report the best results found by the priority rules. SGA-scratch and SGA-modified are the SGAs with rescheduling from scratch and from a modified population, respectively. The results of the two genetic approaches are the average best results of the ten problems of each scenario. The best results were obtained from 10 runs for each problem. The percentage improvement of the SGAs over the priority rules is also shown. Both versions of the SGA outperform the priority rules, and the SGA-modified performs better than SGA-scratch. The relative superiority of the SGA-modified is higher under heavy workload, such as the runs of the 90% and 95% balanced workload mod-

Table 5: The Normalized Results of the Weighted Flow Time Objective in the Stochastic Problems

	Balance Workload					Unbalance Workload				
	75%	80%	85%	90%	95	60%	65%	70%	75%	80%
Priority Rules	3.809	4.562	4.395	4.730	5.442	3.500	3.619	3.650	4.027	4.286
SGA-scratch	3.493(8.3)	4.134(9.4)	4.141(5.8)	4.436(6.2)	5.290(2.8)	3.275(6.4)	3.325(8.1)	3.369(7.7)	3.771(6.3)	3.880(9.5)
SGA-modified	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)

els. This agrees with the implications of Section 3.3. 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. Table 6 shows the results of the other objective functions with respect to different workloads and due date tightnesses. We report only the results of the SGA-modified and the percentage improvement over the priority rules because the results of the priority rules can be calculated from the information provided. The results given are the average best results of the 5 problems of each scenario. The best results are obtained from 5 GA runs for each problem. 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 250 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

Table 6: The Normalized Results of the Stochastic Problems

F	Balance Workload					Unbalance Workload				
	75%	80%	85%	90%	95%	60%	65%	70%	75%	80%
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.

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 job or machine parameters. This result also shows the superiority of the GA for the nonregular objective function.

6 CONCLUSION

This paper extends the previous research on 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. The experimental results show that significant improvement over priority rule approaches was achieved for both deterministic and stochastic JSSPs using a genetic algorithm approach. 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.* [14] 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.

References

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

[2] 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.

[3] 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.

[4] Dorndorf, U. and Pesch, E. "Evolution Based Learning in a Job Shop Scheduling Environment," *Computers Operations Research*, vol. 22, pp. 25-40, 1995.

[5] 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.

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

[7] 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.

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

[9] Goodman, E. D. *An Introduction to GALOPPS*, Technical Report GARAGe95-06-01, Genetic Algorithms Research and Applications Group, Michigan State University, 1995.

[10] 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.

[11] Lin, S.-C., Goodman, E.D., and Punch, W.F., "Investigating Parallel Genetic Algorithms on Job Shop Scheduling Problems," accepted for publication in the Sixth Annual Conference on Evolutionary Programming, 1997.

[12] Morton, T.E., and Pentico, D.W., *Heuristic Scheduling Systems*, John Wiley & Sons, 1993.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] 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.