PERGAMON

Computers & Operations Research 30 (2003) 1087–1102

computers & operations research

www.elsevier.com/locate/dsw

A multi-population genetic algorithm to solve multi-objective scheduling problems for parallel machines

Jeffery K. Cochran*, Shwu-Min Horng, John W. Fowler

Department of Industrial Engineering, Arizona State University, Tempe, AZ 85287-5906, USA

Received 1 October 2000; received in revised form 1 February 2002

Abstract

In this paper we propose a two-stage multi-population genetic algorithm (MPGA) to solve parallel machine scheduling problems with multiple objectives. In the first stage, multiple objectives are combined via the multiplication of the relative measure of each objective. Solutions of the first stage are arranged into several sub-populations, which become the initial populations of the second stage. Each sub-population then evolves separately while an elitist strategy preserves the best individuals of each objective and the best individual of the combined objective. This approach is applied in parallel machine scheduling problems with two objectives: makespan and total weighted tardiness (TWT). The MPGA is compared with a benchmark method, the multi-objective genetic algorithm (MOGA), and shows better results for all of the objectives over a wide range of problems. The MPGA is extended to scheduling problems with three objectives: makespan, TWT, and total weighted completion times (TWC), and also performs better than MOGA.

Scope and purpose

Scheduling and sequencing are important factors to survive in the marketplace. Since scheduling began to be studied at the beginning of this century, numerous papers have been published. Almost all optimize a single objective. Many industries such as aircraft, electronics, semiconductors manufacturing, etc., have tradeoffs in their scheduling problems where multiple objectives need to be considered in order to optimize the overall performance of the system. Optimizing a single objective generally leads to deterioration of another objective. For example, increasing the input rate of product into a system generally leads to higher throughput, but also to increased work-in-process (WIP). Parallel machine scheduling is a Polynomial (NP)-hard problem even for the least complex single objective problem solved. No method is able to generate optimal solutions for the multi-objective case in polynomial time. This limits the quality of design and analysis that can be accomplished in a fixed amount of time. For multi-objective problems, determining a good set of Pareto Front solutions provides maximum information about the optimization. Consider a set of solutions for a problem with multiple objectives. By comparing each solution to every

*Corresponding author. Tel.: +1-480-965-3185; fax: +1-480-965-8692.
E-mail address: j.cochran@asu.edu (J.K. Cochran).

PII: S0305-0548(02)00059-X

other solution, those solutions dominated by any other for all objectives are flagged as inferior. The set of non-inferior individuals is then the Pareto Front solution set. In this paper, an extremely computationally efficient method for determining a good set of Pareto Front solutions in multi-objective parallel machine scheduling problems is presented.

Keywords: Genetic algorithms; Multiple objectives; Parallel machine scheduling

1. Introduction

To study the general case of this problem, many factors must be considered including release times, process times, weights, setup times, and due dates. Three objectives representing the general performance of a manufacturing system are considered in this study. They are minimizing makespan, minimizing total weighted completion times (TWC), and minimizing total weighted tardiness (TWT). Each is defined below:

$$\text{Makespan} = \max(C_1, \ldots, C_n) \quad \text{where } C_j \text{ is the completion time of job } j.$$

$$\text{TWC} = \sum_{j=1}^{n} w_j C_j \quad \text{where } w_j \text{ is the weight (or priority) of job } j.$$

$$\text{TWT} = \sum_{j=1}^{n} w_j T_j \quad \text{where } T_j = \max(0, C_j - d_j), \text{ where } d_j \text{ is the due dates of job } j.$$

Makespan is equivalent to the completion time of the last job leaving the system. Pinedo [1] indicates that a minimum makespan usually implies a high utilization of machines. The utilization for bottleneck or near bottleneck equipment is closely related to the throughput rate of the system. Therefore, reducing makespan should also lead to a higher throughput rate. TWC gives a measure of total holding cost, specifically the inventory costs incurred by the schedule. If all jobs are available at time 0, TWC is the same as total weighted cycle time, perhaps the most common measure of cycle time. When all jobs are not available at time 0, TWC is still highly positively correlated with cycle time, so reducing TWC will typically reduce cycle time. Since cycle time and WIP are directly proportional for a given start rates according to Little's Law (Gross and Harris [2]), reducing TWC will also lead to a reduction in WIP. Finally, TWT is a more general cost function than TWC. When w_j is equal to 1 for all jobs, TWT represents the general case of on-time-delivery. If we can simultaneous control these performance measures, then we can produce a superior schedule.

2. Literature review

The scheduling literature is vast. Only papers focused on multi-objective problems will be reviewed herein. Rosenthal [3] analyzed the principle ideas of multiple objective optimization. Each fundamental idea was examined for strengths and weaknesses. Fry et al. [4] presented a comprehensive review of the published literature on the multiple objective single machine-sequencing problem.

Up to that time, release times of jobs and setups between jobs were not considered. Multiple objectives were either combined using predetermined weights, or one objective was optimized while the other objectives met some threshold criterion. Few of the methods are computationally reasonable when more than 20 jobs are present. Still focusing on the single machine, Lee and Vairaktarakis [5] investigated the complexity of hierarchical scheduling problems. They considered the case of two criteria in which the second criterion is optimized subject to the constraint that the first meets its minimum value. Lee and Jung [6] applied goal programming (GP) to production planning in flexible manufacturing systems to satisfy multiple objectives. The problem is solved using a Pareto approach by assigning different weights to prioritize the objectives. However, when the problem becomes large or the structure is complex, this GP approach becomes impractical.

Some papers have applied the genetic algorithm (GA) heuristic to multiple objective problems. Schaffer [7] developed the vector evaluated genetic algorithm (VEGA) method for finding Pareto optimal solutions of multiple objective optimization problems. In his approach, a population was divided into disjoint sub-populations, where each sub-population optimized its own objective. Murata et al. [8] proposed a multi-objective genetic algorithm (MOGA) and applied it to flowshop scheduling. The selection procedure in MOGA selects individuals with variable weights, which are randomly generated in each generation. The results of MOGA showed better performance than that of Schaffer's VEGA for the objectives of minimizing makespan, total completion times, and total tardiness in which we are interested and is now considered the benchmark in the field.

Hyun et al. [9] developed a new selection scheme in GA, and showed its superiority for multi-objective scheduling problems in assembly lines. Omori et al. [10] studied the solvent extraction process for spent nuclear fuel. One process was modeled as a weighted objective function while the other one became a two-objective function. To optimize these two functions, a GA is used and the results of the GA are compared with that of a random search approach. Also using GAs, Chen et al. [11] studied the radiological worker allocation problem in which multiple constraints are considered. Constraints are classified as hard and soft. Each solution must satisfy the hard constraints and performance of the solution is measured by the violation of soft constraints. The GA approach was compared with conventional optimization techniques such as goal programming and simplex method, and the GA showed superior results.

Other heuristics such as simulated annealing and tabu search have also been studied. Marett and Wright [12] compared these two heuristics for flow shop scheduling problems with multiple objectives. The performance of the methods was compared as the number of objectives increased. Simulated annealing was found to perform better than tabu search as the number of objectives increased. They also mentioned that the complexity of combinatorial problems is strongly influenced by the type of objectives as well as their number. Yim and Lee [13] used Petri nets and heuristic search to solve multi-objective scheduling for flexible manufacturing systems. Pareto optimal solutions were obtained by minimizing the weighted summation of the multiple objectives. Jaszkiewicz [14] combined the genetic algorithm with simulated annealing to solve a nurse-scheduling problem. The maximization of multiple objectives is represented by a single scalarized function, which is the summation of a scalar multiplying the difference between current and previous solutions for each objective. The scalar is greater than one if the objective is improved, or less than one if no improvement is found. Cieniawski et al. [15] applied a GA to solve a groundwater-monitoring problem considering two objectives. The GA approach was compared with simulated annealing on both performance and computational intensity. Simulated annealing relies on a weighted objective function,

which can find only a single point along the trade-off curve of two objectives for each iteration. On the other hand, the GA is able to find a large number points on the trade-off curve in a single iteration.

In this paper, a GA combined with dispatching rules is used to solve the parallel machine scheduling problem. To handle multiple objectives, a two-stage multi-population GA (MPGA) is developed, and its results are compared with that of MOGA, a benchmark GA approach that uses a random weighting scheme for multiple objectives. For the organization of this paper, after the literature review, the details of the two-stage MPGA are described. Using test problems for parallel machine scheduling, six different combination functions of multiple objectives are investigated. Following the description of MPGA, a wide range of test problems with two objectives, makespan and TWT, are used to test it against the benchmark method. The study is extended to scheduling problems with three objectives to be satisfied, namely makespan, TWC, and TWT. Finally, conclusions and future work are presented.

3. The multi-population genetic algorithm (MPGA)

3.1. General approach of MPGA

The two-stage approach we call MPGA is illustrated in Fig. 1. In the first stage, the GA evolves based on the combined objective. The solutions at the end of the first stage are rearranged and divided into sub-populations that then evolve separately. The steps of each stage are described in the following section. Essentially, this approach uses a modification of MOGA in stage one and a modification of VEGA in stage two.

Stage 1—Encoding: For this problem, Fowler et al. [16] developed a method in which a GA is used to assign jobs to machines and then a commonly used strategy, setup avoidance, is used to schedule the individual machines. Each chromosome string, representing a solution, has a number of positions equal to the number of jobs to be scheduled. The number in position one represents the machine that will process job one; the number in position two represents the machine that will process job two; etc. For example, consider a problem with six jobs on three machines. In the feasible solution (3 1 2 3 2 1), job one (position one) is processed on machine three (value of three at first position), and job two is processed on machine one, etc. The completed assignment is thus:

Machine 1: Job 2 and Job 6

Machine 2: Job 3 and Job 5

Machine 3: Job 1 and Job 4

Stage 1—Initialization: Assigning a random integer value between 1 and the number of machines to each digit generates the initial population.

Stage 1—Selection: Selection is an operation to select two parent strings for generating new offspring. Let x_t^i be the ith solution (i between 1 and N, the population size) in the tth generation, and $f(x_t^i)$ be the performance measure of solution x_t^i. Each solution x_t^i is selected as a parent string

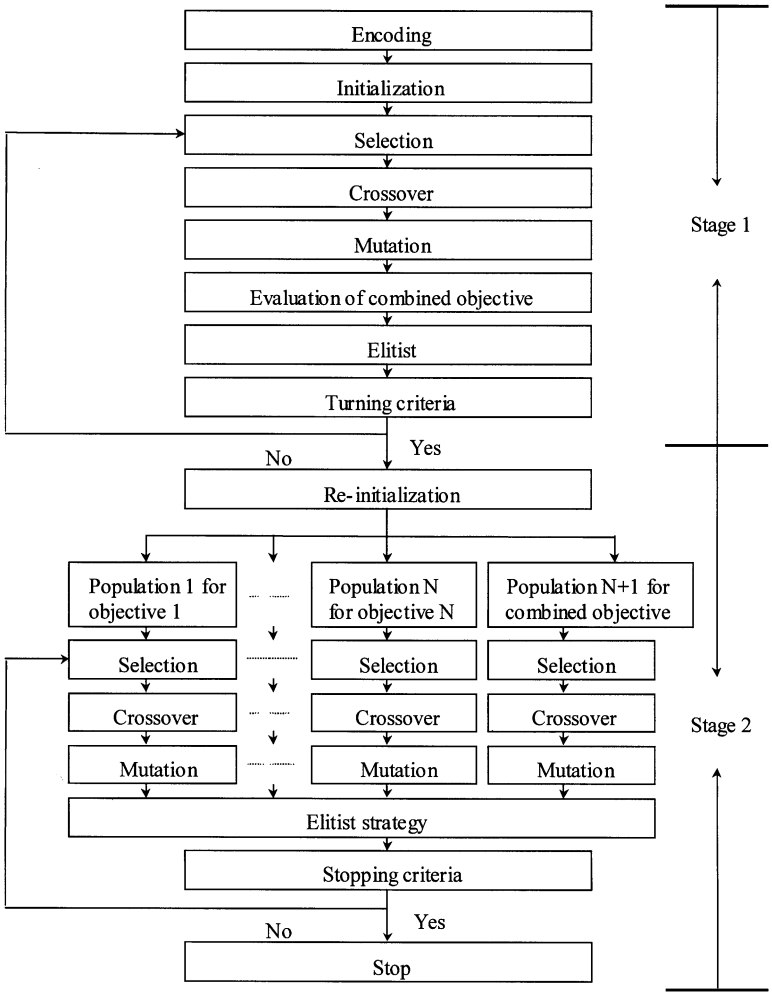

Fig. 1. A two-stage multi-population genetic algorithm (MPGA) for multi-objective scheduling problems.

according to the selection probability $P(x_t^i)$. The following method is used:

$$P(x_t^i) = \frac{[f_t^w - f(x_t^i)]^2}{\sum_{k=1}^{N} [f_t^w - f(x_t^k)]^2},$$

where f_t^w is the worst, maximum value of objective at generation t.

Stage 1—Crossover: Crossover is an operation where two parent strings exchange parts of their corresponding chromosomes. For example, two child strings, (132312) and (212331) are selected according to pre-defined crossover probabilities. A crossover point (from 1 to 5 in this example) is randomly selected (assume 2 is selected), and then two new strings ((132331) and (212312)) are created.

Stage 1—Mutation: Mutation is an operation that changes one digit at a time. A digit is selected according to pre-defined mutation probabilities and replaced with a different number. For example,

digit five in string (132312) is selected and 2 is randomly selected as the number to be inserted, thus this string becomes (132322).

Stage 1—Evaluation of the combined objective: The MOGA approach combines the objectives by weighting to form a single function for optimization. Assuming N objectives are to be optimized, the objective function of MOGA is formulated as

$$f_{t,j}(x) = \sum_{i=1}^{n} w_{t,i} f_{t,i,j}(x) \quad i = 1,\ldots,N, \;\; j = 1,\ldots,P$$

where P is the population size, $f_{t,j}(x)$ is the combined function at generation t of string j, $w_{t,i}$ is the weight of function i at generation t, and $f_{t,i,j}(x)$ is the ith objective function of string j at generation t. Note that $w_{t,i}$ is randomly generated at each generation. A new function for combining multiple objectives is proposed in this study and is discussed in the next section.

Stage 1—Elitist strategy: The best solution of each objective and the best one of the combined objective function are preserved in each generation. For each generation, if the best solution is worst than the preserved one, a randomly selected string is replaced with the preserved solution.

Stage 1—Turning criterion: After a certain number of generations, the algorithm switches to the next stage. In stage 2, the populations of the solution from stage 1 will be rearranged based on their performance with respect to each objective.

Stage 2—Re-initialization: Assume N objectives to be optimized, as discussed above. The solutions from the first stage are rearranged and $N+1$ sub-populations are created to now evolve separately. The first through Nth sub-populations are for the N objectives, and the $(N+1)$st sub-population is for the combined objective.

Stage 2—Selection, crossover, and mutation: The same selection, crossover, and mutation procedures used in stage 1 are applied in each sub-population.

Stage 2—Elitist strategy: While each sub-population evolves separately, the elitist strategy searches for the best solution of each objective and combined objective across all sub-populations. $N+1$ solutions are stored and replace the worst of each objective and the combined objective in the next generation.

Stage 2—Stopping criteria: Test runs indicate that our algorithm does not show significant improvement after 4,000 generations. However, there could be searches with unusual errors due to randomness, so to be conservative, a larger generation number should be used as the stopping criteria. In addition, many GA studies in scheduling from the literature use 5,000 generations to stop the algorithm. Therefore, generation 5000 will be used in this study as the stopping criteria.

3.2. Important issues of MPGA

Several important issues that affect the performance of MPGA will be discussed in this section: a new function for combining all objectives in stage 1, the GA parameter (population size, crossover probability, and mutation probability) settings selected, and the criteria to turn from stage 1 to stage 2.

MOGA has been shown to produce good solutions in the job shop scheduling problem with three objectives, but it does not account for the magnitude of each objective. For example, consider scheduling 20 jobs on a single machine, when each job needs an average processing time of 50 and all jobs are ready at the beginning. The makespan is about 1,000 while the total completion

Table 1
Six objective functions of multiple objective in scheduling problems

Case	Objective function
1	$f_{t,j}(x) = \sum_{i=1}^{n} f_{t,i,j}(x)$
2	$f_{t,j}(x) = \sum_{i=1}^{n} w_{t,i} f_{t,i,j}(x)$
3	$f_{t,j}(x) = \sum_{i=1}^{n} (f_{t,i,j}(x)/f^*_{t,i,j}(x))$
4	$f_{t,j}(x) = \sum_{i=1}^{n} w_{t,i}(f_{t,i,j}(x)/f^*_{t,i,j}(x))$
5	$f_{t,j}(x) = \prod_{i=1}^{n} (f_{t,i,j}(x)/f^*_{t,i,j}(x))$
6	$f_{t,j}(x) = \sum_{i=1}^{n} (f_{t,i,j}(x) - f^*_{t,i,j}(x))/f^*_{t,i,j}(x)$

Note: $f^*_{t,i}(x)$ is the best solution of objective i at generation t.

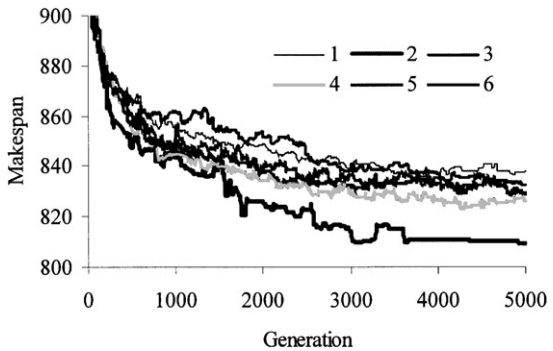

Fig. 2. Comparison Pareto Front average of makespan (lines 1 to 6 are the average Pareto Fronts of functions 1 to 6, respectively, in Table 1 evolving through generations).

time is about 10,000. Single summation of these two objectives will favor the one with the larger value. To correct this discrimination, a relative measure is proposed. The relative measure uses the performance of each objective divided by the best value of each objective at the current generation. Table 1 lists six different objective functions that we have studied. The first is the summation of all objectives with equal weights. The second one is from MOGA with random weights. The third and fourth are the summation and randomly weighted summation of relative measures, respectively. The fifth one is the product of relative measures. The last one measures the total percentage difference between each objective measure and the best value of each objective.

To study the six objective functions shown in Table 1, we used a parallel machine scheduling problem with two objectives, makespan and TWT. The GA parameters are set to be 20 for population size, 0.6 for crossover probability, and 0.01 for mutation probability along with an elitist strategy preserving best individual of makespan, TWT, and combined objective. These parameter settings are different than Murata's algorithm, but they produce better solutions than the original MOGA according to our study. Therefore, these parameter settings are used throughout this paper. Figs. 2 and 3 show the average Pareto Front of each of the six objective forms evolving through generations. MOGA (function 2) produces the best solutions for makespan at the end of the search, but functions

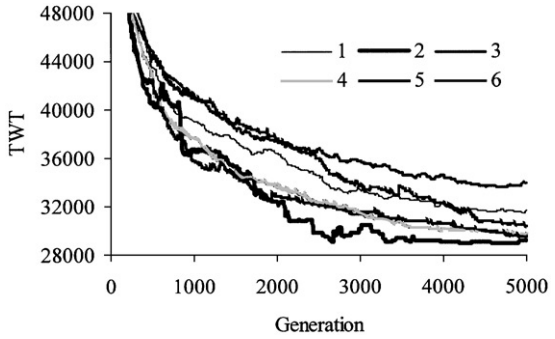

Fig. 3. Comparison Pareto Front average of TWT (lines 1 to 6 are the average Pareto Fronts of functions 1 to 6, respectively, in Table 1 evolving through generations).

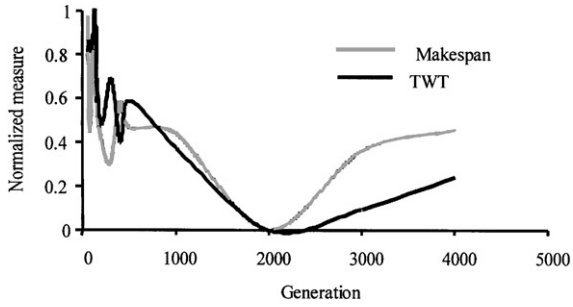

Fig. 4. Performance of different turn points as optimizing makespan and TWT.

4 and 5 produce better solutions at the objective of TWT before half way through the generations, and good results overall. We have found that improved solutions are obtained if the solutions from function 4 or 5 obtained halfway through the generations are used as the starting points of another algorithm. This is the motivation of the two-stage MPGA. It turns out that using VEGA, modified to use our objective function, is an excellent choice for the second half of optimization. Since function 5 performs better than function 4, it will be used at first stage of MPGA throughout the study.

To test this hypothesis a test problem with two objectives is selected where the best six solutions of makespan form the first sub-population, the best six solutions of TWT form the second sub-population, and the best eight solutions of the combined objective form the third sub-population. This combination of sub-population sizes $(6, 6, 8)$ in stage 2 worked well throughout our studies. These three sub-populations are then evolved separately to obtain final solutions. Each case with a different turn point (number of generations) is simulated 30 times to obtain averages. Fig. 4 shows the performance of MPGA in terms of makespan and TWT of the final solutions with different values for the turning criteria. The normalized measure is over the range of $[0, 1]$ and calculated based on the following equation:

$$\textit{Normalized measure} = (\textit{value} - \textit{minimum value})/(\textit{maximum value} - \textit{minimum value}).$$

Fig. 4 indicates that the turning point between MOGA and VEGA should be around 50% of the completed evolution generations in order to have the best performance. We will show below that

Table 2
Characteristics of test problem

Factors	Levels	Description
Range of weights	Narrow	$[1,10]$.
	Wide	$[1,20]$.
Range of due dates	Narrow	Release time $+ [-1,2] \times$ total process time.
	Wide	Release time $+ [-2,4] \times$ total process time.
Ratio ($\bar{p}/\bar{s}$)	High	$50/10$, $p = 50 + [-9,9]$, $s = 2 * [3,7]$.
	Moderate	$30/30$, $p = 30 + [-9,9]$, $s = 6 * [3,7]$.
	Low	$10/50$, $p = 10 + [-9,9]$, $s = 10 * [3,7]$.
WIP status	High	All jobs are ready at time 0.
	Moderate	50% of jobs are ready at time 0 and the others are ready at time $[0,720]$.
	Low	All jobs are ready at time $[0,720]$.

this 50/50 strategy, combined with the new objective function form we use in both stages, produces solutions significantly improved over MOGA alone, which has already been shown in the literature to be superior to VEGA alone.

4. Computational results

4.1. Test problems

To show how the MPGA performs on the parallel machine scheduling problem with multiple objectives, the test problems of Fowler et al. [16] are used as shown in Table 2. The weights of jobs are random numbers in the interval $[1,20]$ at the wide range, and in the interval $[1,10]$ at the narrow range. The wide range of due dates of jobs are the release time of a job plus a random number in the interval $[-2,4]$ times the summation of its process time and average setup times. Replacing the interval $[-2,4]$ with $[-1,2]$ creates due dates with a narrow range. Negative numbers for due dates indicate that jobs are already late upon entering the system. Often, this is the case in real situations and is not commonly considered in scheduling optimization methods. The process times and setup times of jobs are combined into one factor that is generated at three levels. Ratios of average process times to average setup times are equal to $5, 1$, and $\frac{1}{5}$, representing high, moderate, and low levels, respectively. The different level of ratios represents the impact of the magnitude of the setup time on total process times. For WIP status equal to high, all jobs are ready at time 0. When WIP is moderate, 50% of the jobs are ready at time 0, and the other 50% of the jobs each become available equally likely during the time interval $[0,720]$. All jobs are equally likely during the time interval $[0,720]$ when WIP is low. A total of 100 jobs for each problem instance need to be scheduled on 5 parallel machines with the objectives of minimizing makespan, and TWT.

4.2. Performance measures

The purpose of multiple objective problems is to obtain Pareto Front solutions. Shaffer [7] and many other researchers use Pareto Front solutions as a quantitative measure of the performance of

Table 3
Example to calculate the quantitative and qualitative measures of two algorithms

Algorithms	A	B
Number of solutions of Pareto Front	5	8
Number of solutions of combined Pareto Front	6	6
Number of solutions form the combined Pareto Front	2	4
Quantitative measure	5	8
Qualitative measure	2/6	4/6

the algorithms studied. Hyun et al. [9] developed another measure to compare two algorithms for their quality of solutions. Suppose algorithm A obtains a set of N_1 Pareto Front solutions, algorithm B obtains a set of N_2 Pareto Front solutions, and the combined Pareto Front from the two algorithms contains N_c solutions, where N_c is larger than or equal to the minimum of N_1 and N_2, and less than or equal to N_1 plus N_2. N_1 and N_2 represent the quantitative performance of algorithms A and B, respectively. N_1/N_c and N_2/N_c are then used to indicate the performance of algorithms A and B, respectively, in a qualitative sense. For example, assume algorithm A obtains 5 Pareto Front solutions and algorithm B obtains 8 Pareto Front solutions. Further, assume that combining these two sets of solutions produces a Pareto Front with 6 solutions; 2 from algorithm A and 4 from algorithm B. Table 3 shows the quantitative and qualitative measures of both algorithms. We will calculate and report both quantitative and qualitative measures in our algorithm comparison below.

4.3. MPGA vs. MOGA in parallel machine scheduling problem with two objectives

The test problems in Table 2 are used to compare the performance of MPGA to MOGA in parallel machine scheduling problems with respect to two objectives, makespan, and TWT. The four factors contain 36 ($2 \times 2 \times 3 \times 3$) test problems. Each problem is iterated 10 times with different seed numbers to create 10 instances, so a total of 360 instances (test problems) are generated. For each instance, an average of 10 runs of MOGA and MPGA is calculated. The results are shown in Table 4. Each number in the table is the average of 100 (10×10) simulations. For the quantitative measure described in the previous section, MPGA outperforms MOGA for all test problems. For the qualitative measure, MPGA is better than MOGA for 32 (89%) out of the 36 test problems. In Table 4, the "Win" column counts the number of winning instances for each test problem. MPGA wins at least 5 out of 10 instances for all 36 test problems in quantitative measure, and 31 out of 36 test problems in qualitative measure. On average, MPGA produces 8.97 Pareto Front solutions while MOGA only has 6.56. For the combined Pareto Front from both algorithms, average 60.7% are from MPGA and 39.3% are from MOGA. These results indicate that MPGA not only produces more Pareto Front solutions, but also better solutions than MOGA.

To have a better understanding of the few cases (4 out of 36) when MOGA outperforms MPGA qualitatively, results are presented by the factors of the test problems and shown in Table 5. For the first two factors, range of weights and range of due dates, both algorithms produce better solutions as the ranges of these two factors become larger. This behavior is predictable since neither factor affects Makespan, but they have a strong effect on TWT. As both ranges get larger, TWT will be very sensitive to small changes in job schedules. Although the effects are not significant, MPGA seems

Table 4
Comparison of MPGA and MOGA at quantitative and qualitative measures

					Quantitative			Qualitative				
Prob #	Weight	Due	Ratio	WIP	MPGA	MOGA	Win	MPGA	MOGA	MPGA (%)	MOGA (%)	Win
1	N	N	H	H	11.88	9.39	9.5	6.62	5.16	56.2	43.8	5.5
2	N	N	H	M	11.98	9.52	9.5	8.18	4.15	66.3	33.7	8.5
3	N	N	H	L	10.69	8.37	8.5	7.90	2.46	76.3	23.7	10
4	N	N	M	H	11.74	8.79	10	6.02	5.38	52.8	47.2	6
5	N	N	M	M	5.70	4.55	8	3.51	1.87	65.2	34.8	8
6	N	N	M	L	6.95	5.40	9	4.41	2.32	65.5	34.5	9
7	N	N	L	H	12.56	8.64	9	5.75	5.55	50.9	49.1	3
8	N	N	L	M	1.82	1.22	9.5	0.61	0.72	45.9	54.1	3
9	N	N	L	L	2.93	1.85	8	1.17	1.06	52.5	47.5	6
10	N	W	H	H	11.99	9.70	8.5	7.31	4.74	60.7	39.3	7.5
11	N	W	H	M	14.05	9.45	10	9.53	4.04	70.2	29.8	9
12	N	W	H	L	10.8	8.92	9	7.91	2.95	72.8	27.2	10
13	N	W	M	H	11.4	9.15	9.5	5.99	5.38	52.7	47.3	6
14	N	W	M	M	6.67	4.76	8	3.92	2.01	66.1	33.9	8
15	N	W	M	L	7.88	5.29	10	5.02	2.46	67.1	32.9	8
16	N	W	L	H	12.5	9.53	10	5.67	6.13	48.1	51.9	4
17	N	W	L	M	2.06	1.27	9	0.81	0.70	53.6	46.4	6
18	N	W	L	L	4.33	1.94	9.5	1.48	1.28	53.6	46.4	5
19	W	N	H	H	12.15	8.86	10	6.61	4.78	58.0	42.0	5
20	W	N	H	M	13.32	9.38	9.5	9.69	3.70	72.4	27.6	9.5
21	W	N	H	L	11.49	8.79	10	7.14	3.67	66.0	34.0	8.5
22	W	N	M	H	11.91	8.71	10	6.73	4.53	59.8	40.2	6
23	W	N	M	M	5.74	4.77	5.5	3.68	1.38	72.7	27.3	9
24	W	N	M	L	8.17	5.35	10	4.52	2.69	62.7	37.3	7
25	W	N	L	H	12.57	9.30	9.5	5.97	6.00	49.9	50.1	4.5
26	W	N	L	M	1.59	1.27	8	0.69	0.66	51.1	48.9	5
27	W	N	L	L	3.30	1.99	9	1.31	1.13	53.7	46.3	6
28	W	W	H	H	12.44	9.31	8.5	6.91	4.90	58.5	41.5	7
29	W	W	H	M	13.81	9.42	9.5	10.67	3.23	76.8	23.2	10
30	W	W	H	L	11.85	9.24	9	8.87	3.04	74.5	25.5	10
31	W	W	M	H	11.69	8.57	9	6.53	4.66	58.4	41.6	6
32	W	W	M	M	7.46	5.15	6	5.00	1.77	73.9	26.1	9.5
33	W	W	M	L	8.46	5.55	10	4.66	2.86	62.0	38.0	7
34	W	W	L	H	12.88	9.58	9.5	6.09	6.00	50.4	49.6	5
35	W	W	L	M	1.87	1.29	8.5	0.67	0.83	44.7	55.3	4.5
36	W	W	L	L	4.18	2.00	10	1.75	1.04	62.7	37.3	5.5
Avg.					8.97	6.56	9.1	5.26	3.20	60.7	39.3	6.9

Note: the column of "Win" represents how many instances (out of 10) MPGA performs better than MOGA.

to performs better when the range of weights is larger and the range of due dates is smaller. The other two factors, Ratio and WIP, show significant effects on the performance of both algorithms. When the ratio of average process times to average setup times becomes lower, jobs of the same kind tend to be grouped together in order to reduce setup times. With this condition constraining the

Table 5
Comparison of MPGA and MOGA based on the individual factor of test problems

				Quantitative			Qualitative				
Weight	Due	Ratio	WIP	MPGA	MOGA	Win	MPGA	MOGA	MPGA (%)	MOGA (%)	Win
N				8.77	6.54	164.5	5.10	3.24	59.8	40.2	122.5
W				9.16	6.59	161.5	5.42	3.16	61.6	38.4	125
	N			8.69	6.45	162.5	5.03	3.18	59.9	40.1	119.5
	W			9.24	6.67	163.5	5.49	3.22	61.5	38.5	128
		H		12.20	9.20	111.5	8.11	3.90	67.4	32.6	100.5
		M		8.65	6.34	105	5.00	3.11	63.2	36.8	89.5
		L		6.05	4.16	109.5	2.66	2.59	51.4	48.6	57.5
			H	12.14	9.13	113	6.35	5.27	54.7	45.3	65.5
			M	7.17	5.17	101	4.75	2.09	63.2	36.8	90
			L	7.59	5.39	112	4.68	2.25	64.1	35.9	92

Note: the column of "Win" represents how many instances (out of 180 for factors Weight and Due, and out of 120 for factors Ratio and WIP) MPGA performs better than MOGA.

Table 6
Comparison of MPGA and MOGA based on two significant factors (ratio of average process times to average setup times and WIP status) of test problems

		Quantitative			Qualitative				
Ratio	WIP	MPGA	MOGA	Win	MPGA	MOGA	MPGA (%)	MOGA (%)	Win
H	H	12.12	9.32	36.5	6.86	4.90	58.4	41.6	25
H	M	13.29	9.44	38.5	9.52	3.78	71.4	28.6	37
H	L	11.21	8.83	36.5	7.96	3.03	72.4	27.6	38.5
M	H	11.69	8.81	38.5	6.32	4.99	55.9	44.1	24
M	M	6.39	4.81	27.5	4.03	1.76	69.5	30.5	34.5
M	L	7.87	5.40	39	4.65	2.58	64.3	35.7	31
L	H	12.63	9.26	38	5.87	5.92	49.8	50.2	16.5
L	M	1.84	1.26	35.0	0.70	0.73	48.8	51.2	18.5
L	L	3.69	1.95	36.5	1.43	1.13	55.6	44.4	22.5
Total				326					247.5

Note: the column of "Win" represents how many instances (out of 40) MPGA performs better than MOGA.

searching space, the number of Pareto Front solutions is reduced and the qualitative performance of MPGA deteriorates. For the factor of WIP status, except when it is at the high level (all jobs are ready at the beginning), MPGA outperforms MOGA.

Table 6 shows the results organized on levels of the factors ratio and WIP. Both algorithms produce few good solutions when the ratio is low and WIP is moderate or low. By examining each simulation of this circumstance, it can be seen that few solutions, usually less than three, dominate the Pareto Front. These solutions have the same measure of Makespan while improving the other objective, namely reducing TWT. Makespan can be easily minimized by sequencing jobs according to a first-in-first-out (FIFO) policy, because either 50% or all the jobs arrive in the interval

Table 7
Comparison of MPGA and MOGA for the scheduling problems with three objectives, Makespan, TWC, and TWT

Quantitative		Qualitative			
MPGA	MOGA	MPGA	MOGA	MPGA (%)	MOGA (%)
13.94	10.65	8.78	4.34	66.9	33.1

of [0, 720]. This can also explain why MOGA performs slightly, but not significantly, better than MPGA for these two scenarios. The sub-populations of MPGA for Makespan find the near optimal solutions at an early generation and do not improve at the latter generations. Compared to the strategy of MPGA, MOGA uses all its time searching all objectives at each generation. The efficiency of MPGA is reduced if any of the objectives is comparatively simple and easy to be found. For the entire test problem instances (360), MPGA outperforms MOGA for 326 instances quantitatively, and 248 instances qualitatively. CPU times of both algorithms are compared and no significant difference can be found.

4.4. MPGA vs. MOGA in parallel machine scheduling problems with three objectives

Now let's raise the challenge to a higher level by comparing MPGA with MOGA in scheduling problems with three objectives, Makespan, TWT, and total weighted completion time (TWC). The center point of the four factors described in Table 2 is used as the test problem. Ten instances are generated using different seed numbers. Each algorithm is simulated 10 times for each instance to obtain the averages. The parameters for both algorithms remain the same except for population size. For MPGA, six additional populations are added to the program for the objective of TWC, and the total number of populations is 26 ($6 \times 3 + 8$). The same number of populations is then used by MOGA. The results are shown in Table 7. Again, MPGA outperforms MOGA both quantitatively and qualitatively. MPGA's qualitative performance increases slightly from 60.7% of two objective-scheduling problems to 66.9% of three-objective scheduling problems.

To help the reader see graphically the difference in performance of the two algorithms, Fig. 5 illustrates the comparison of MPGA with MOGA in one simulation of one instance. Two-dimensional figures are used to represent the Pareto Front solutions of both algorithms. For makespan-TWC and makespan-TWC comparisons, MPGA clearly produces better solutions than MOGA. The TWT–TWC solutions of both algorithms do not follow the concave-shape curves like the other two because of the high correlation between TWC and TWT. In spite of this irregular shape, MPGA still outperforms MOGA in the TWC–TWT comparison.

5. Conclusions and future work

Solving an NP-hard scheduling problem with only one objective is a difficult task. Heuristics have been developed to get near optimal solutions. Adding more objectives obviously makes this problem more difficult to solve, particularly for the size and high complexity of the problems studied in this paper.

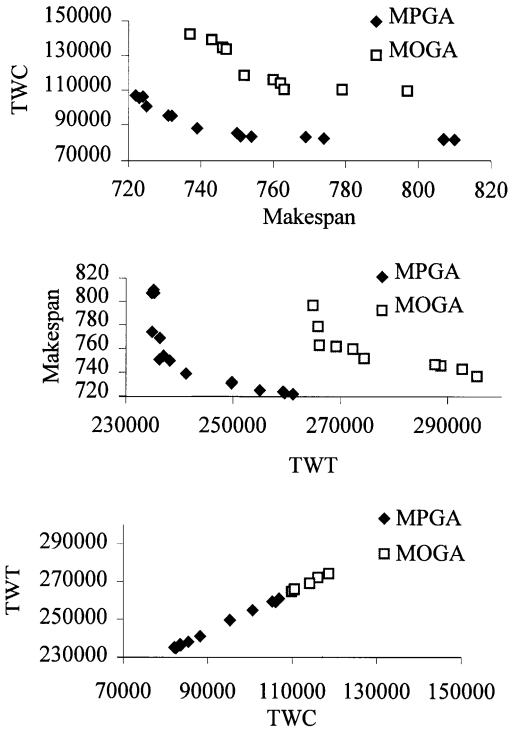

Fig. 5. Comparison of MPGA with MOGA.

There is now a new methodology for such problems, the two-stage multiple population genetic algorithm (MPGA) created by improving and combining two heuristics, the vector evaluation genetic algorithm (VEGA) and the multiple objective genetic algorithm (MOGA). MPGA uses a better overall form of the objective function for combining the multiple objectives than other studies, and we have learned that using a modified MOGA followed by a modified VEGA (turning between them at about halfway through the optimizing generations) produces solutions superior to either alone at no expense in computer run-time.

In the first stage of MPGA, all of the objectives are combined into a single objective so that the algorithm can converge quickly to good solutions with respect to all the objectives. The solutions of the first stage are then divided into sub-populations that evolve separately. The solutions for each objective are improved within the individual sub-populations while another sub-population contains solutions satisfying the combined objective. Computational results show that the two-stage MPGA outperforms MOGA in most of the test problems with two objectives. An extended study in the three-objective parallel machine scheduling problem also indicates the superiority of the two-stage MPGA to the MOGA. Although the MPGA approach in this study is used to solve scheduling problems, it can be easily applied to other optimization problems by changing the encoding scheme of the GA.

The two-stage MPGA shows good results compared to that of MOGA, considered the best method to date, but additional improvements may be possible. Both algorithms produce many unwanted solutions that are dominated by other solutions. A more advanced genetic algorithm might be developed

to take this issue into consideration. In addition, high variance among iterations of the genetic algorithm is observed, but this is a problem for every GA researcher. Reducing the variance among the iterations of the genetic algorithm is another research direction so that more of the solutions of any iteration of the genetic algorithm will fall within a desirable range.

Acknowledgements

The authors gratefully acknowledge the support of the National Science Foundation (Grant DMI-9713750) and the Semiconductor Research Corporation (Contract 97-EJ-492).

References

[1] Pinedo M. Scheduling: theory, algorithms, and systems. Englewood Cliffs, NJ: Prentice-Hall, 1995.
[2] Gross D, Harris CM. Fundamentals of queueing theory, 3rd ed. New York: Wiley, 1988.
[3] Rosenthal RE. Concepts, theory, and techniques—principle of multiobjective optimization. Decision Science 1985;16:133–52.
[4] Fry TD, Armstrong RD, Lewis H. A framework for single machine multiple objective sequencing research. Omega 1989;17:594–607.
[5] Lee C-Y, Vairaktarakis GL. Complexity of single machine hierarchical scheduling: a survey. In: Pardalos PM, editor. Complexity in numerical optimization. Singapore: World Scientific Publishing, 1993. p. 269–98.
[6] Lee SM, Jung H-J. A multi-objective production planning model in a flexible manufacturing environment. International Journal of Production Research 1989;27:1981–92.
[7] Schaffer JD. Multiple objective optimization with vector evaluated genetic algorithms. Proceedings of the First ICGA, 1985. p. 93–100.
[8] Murata T, Ishibuchi H, Tanaka H. Multi-objective genetic algorithm and its application to flowshop scheduling. Computers and Industrial Engineering 1996;30:957–68.
[9] Hyun CJ, Kim Y, Kin YK. A genetic algorithm for multiple objective sequencing problems in mixed model assembly. Computers & Operations Research 1998;25:675–90.
[10] Omori R, Sakakibara Y, Suzuki A. Applications of genetic algorithms to optimization problems in the solvent extraction process for spent nuclear fuel. Nuclear Technology 1997;118:26–31.
[11] Chen Y, Narita M, Tsuji M, Sa S. A genetic algorithm approach to optimization for the radiological worker allocation problem. Health Physics 1996;70:180–6.
[12] Marett R, Wright M. A comparison of neighborhood search techniques for multi-objective combinatorial problems. Computers and Operations Research 1996;23:465–83.
[13] Yim SJ, Lee DY. Multiple objective scheduling for flexible manufacturing systems using petri nets and heuristic search. Proceedings IEEE International Conference on Systems, Man, and Cybernetics 1996;4:2984–9.
[14] Jaszkiewicz A. A metaheuristic approach to multiple objective nurse scheduling. Foundations of Computing and Decision Sciences 1997;22:169–83.
[15] Cieniawski SE, Eheart JW, Ranjithan S. Using genetic algorithms to solve a multiobjective groundwater monitoring problem. Water Resources Research 1995;31:399–409.
[16] Fowler JW, Horng S-M, Cochran JK. A hybridized genetic algorithm to solve parallel machine problem scheduling problem with sequence dependent setups. International Journal of Manufacturing Technology and Management 2000;1:3–4.

Jeffery K. Cochran is Professor of Industrial Engineering at Arizona State University. He received his Ph.D. in Operations Research from Purdue University. His primary research interests are optimization of stochastic models in high technology production systems, next-generation discrete event simulation tools, and transport system and supply chain logistics. Dr. Cochran is the author of over one hundred scholarly publications, and is a Professional Engineer.

Shwu-Min Horng has been a Project Manager and a Senior Supply Chain Analyst for Motorola, Inc. for two years. He received his Ph.D. in the area of Operations Research in the Department of Industrial Engineering in 1998. Dr. Horng's research interests include optimization, system simulation and analysis, supply chain management, and artificial intelligence.

John W. Fowler is an Associate Professor in the Industrial Engineering Department at Arizona State University. He received his Ph.D. in Industrial Engineering from Texas A&M University. His research interests include modeling, analysis, and control of semiconductor manufacturing systems. Dr. Fowler is the co-director of the Modeling and Analysis of Semiconductor Manufacturing Laboratory at ASU. He is also an Area Editor for *SIMULATION*: *Transactions of the Society for Modeling and Simulation International—Applications* and Associate Editor of *IEEE Transactions on Electronics Packaging Manufacturing*.