International Journal of Operations Research

International Journal of Operations Research Vol. 4, No. 1, 1−10 (2007)

Effectiveness of Adaptive Crossover Procedures for a Genetic Algorithm to Schedule Unrelated Parallel Machines with Setups

Patricia A. Randall[1,*] and Mary E. Kurz[2]

[1]Department of Industrial Engineering, Clemson University, 110 Freeman Hall, Clemson, SC 29634, USA

[2]Department of Industrial Engineering, Clemson University, 104A Freeman Hall

Received May 2006; *Revised August 2006*; *Accepted September 2006*

Abstract—The unrelated parallel machine scheduling problem, in its most general form, is applicable to many manufacturing and service environments. This problem requires the scheduling of a group of independent jobs on unrelated parallel machines as well as the sequencing of the jobs on each individual machine. In this paper, we propose a genetic algorithm with adaptive crossover selection to schedule independent jobs on unrelated parallel machines to minimize total tardiness. Each job has a unique due date, machine-dependent processing times, and sequence-dependent setup times. Three general adaptive crossover selection schemes will be compared with a traditional genetic algorithm and tabu search for large-scale problems (up to 200 jobs and 20 machines). The adaptive genetic algorithm with the tournament selection scheme is shown to outperform all other heuristics with respect to solution quality although it does require more solving time than many of the other heuristics.

Keywords—Unrelated parallel machine scheduling, Genetic algorithms, Sequence-dependent setups

1. INTRODUCTION

This paper presents three types of adaptive crossover selection schemes within a traditional genetic algorithm to schedule unrelated parallel machines. In the unrelated parallel machine scheduling problem, denoted $R||\bullet$, a group of independent jobs are available to be scheduled on multiple machines at a single stage. Machining characteristics such as processing times and setup times are unique to each machine and are unrelated to machining characteristics of other machines. In the most general case of this problem, all jobs are ready to be processed at time zero and all machines are ready to process jobs at time zero. Each machine is capable of processing every job. A job must be processed completely by one machine and job preemption is not allowed. Although completion time objectives are dominant in early scheduling literature, an emphasis on satisfying deadlines in industry has caused the focus of recent unrelated parallel machine scheduling literature to shift towards due date-related objectives such as minimizing the total tardiness of the jobs, denoted $\sum T_j$, and occasionally the total earliness of jobs as well, denoted $\sum E_j$. For a review of scheduling notation, see Graham et al. (1979).

In our problem, N independent jobs are scheduled on M unrelated machines at a single stage. Each job j has a unique due date d_j, a processing time p_{jk} when it is processed on machine k, and a setup time s_{ijk} when it is preceded by job i on machine k. The completion time of job j is C_j and the tardiness of job j is $T_j = \max\{0, C_j - d_j\}$. A traditional genetic algorithm and tabu search will be compared with three adaptive crossover selection schemes (regulatory gene, roulette wheel selection, and tournament selection).

2. LITERATURE REVIEW

While linear programming models are not able to solve problems of large sizes for unrelated parallel machine scheduling in reasonable time, they give optimal solutions that can be used to test the quality of other non-optimal solution heuristics. Alidaee and Panwalker (1993), by employing a common due date for all jobs, are able to reduce $R||\sum E_j + \sum T_j$ to a transportation problem that can be solved in polynomial time. Cheng et al. (1996) reduce the above problem to an assignment problem with polynomial complexity through the use of a processing time compression cost. Logendran and Subur (2004) develop a mixed integer programming (MIP) model for $R||\sum T_j$ where jobs have release times and machines have availability times. There is also the possibility of splitting a job into two lots that can be processed on different machines. Liaw et al. (2003) develop a branch-and-bound algorithm that incorporates a lower bound found by solving a $R||\sum E_j + \sum T_j$ -based assignment problem.

[*]Corresponding author's email: patricia.randall@alumni.clemson.edu

Search heuristics tend to be more effective for larger problem sizes than optimal methods because they are able to quickly consider many different solutions. While an optimal solution is never guaranteed, many search heuristics consistently yield optimal or near-optimal solutions. Randall and Kurz (2005) show that a random keys genetic algorithm finds optimal and near-optimal solutions for $R\|\sum T_j$ where jobs have sequence-ependent setups.

Bank and Werner (2001) first assign each job to a machine and then compare heuristics to schedule each machine for $R\|\sum E_j + \sum T_j$ with a common due date and job release times. Neighborhood search, iterative improvement, multistart procedures, simulated annealing, and threshold accepting heuristics are compared against constructive heuristics. Kim et al. (2002) develop a simulated annealing heuristic for $R\|\sum T_j$ with sequence-dependent setups. A job is assumed to be composed of several items, each of which can be processed on a different machine.

Several papers have applied genetic algorithms (GAs) to the parallel machine scheduling problem. Van Hop and Nagarur (2004) apply GAs to the optimization of printed circuit board operations to group similar boards, balance machines, and reduce setups. Cheng et al. (1995) develop a job partitioning chromosome structure within their GA to minimize maximum weighted absolute lateness under a common due date. Luu et al. (2002) sequence batches on parallel machines with a hybrid GA that includes an Earliest Due Date-Greedy insertion method. Jou (2005) develops a genetic algorithm to schedule parallel flow shop machines where jobs are queued in a bottleneck stage. Cochran et al. (2003) develop a two-state multi-population GA with the multiple objectives of minimizing the makespan, minimizing the total weighted tardiness, and minimizing the total weighted completion times.

Min and Cheng (1999) and Kurz and Askin (2001) use GAs to minimize the makespan. After comparing a GA with a slicing heuristic, a multiple MULTI-FIT heuristic, and an insertion heuristic under the constraints of job release times and sequence-dependent setups, Kurz and Askin conclude that the random keys chromosome representation (Bean, 1994) could yield better solutions than a traditional binary representation. Kurz and Askin (2004) develop a GA with the random keys chromosome structure to schedule flexible flowlines with sequence-dependent setups to minimize the completion time.

Norman and Bean (1999) use the random keys chromosome representation to schedule 360 jobs on two parallel machines in an automotive plant with job release times and sequence-dependent setups to minimize the total tardiness. Sivrikaya-Serifoglu and Ulusoy (1999) minimize total earliness and total tardiness for the parallel machine scheduling problem with job release times and sequence-dependent setups through the use of a GA employing their MCUOX crossover operator. Glass et al. (1994) compare a GA with a descent algorithm, simulated annealing, and tabu search for $R\|C_{\max}$ where $C_{\max}$ is the makespan.

Several papers have developed adaptive genetic algorithms (AGAs). Davis (1989) develops an AGA with variable rates for both crossover and mutation operators by tracking the lineage of chromosomes through several generations. Chromosomes with superior ancestors are selected for the majority of crossover and mutation operations performed. Julstrom (1995) expanded the work of Davis to the adaptive operator probabilities mechanism that tracks the ancestry of both chromosomes and crossover and mutation operators. Van Hop and Tabucanon (2005) adjust operator rates by comparing the number of chromosomes that each operator should theoretically create and the number of chromosomes that each operator did create. These ratios of theoretical to actual chromosomes determine the adjusted rates for the operators.

Tuson and Ross (1996) develop an AGA with a cost-based operator rate adaptation mechanism that adjusts rates at a fixed interval. The operator with the largest benefit-to-cost ratio receives the largest operator rate while the operator with the smallest benefit-to-cost ratio receives the smallest operator rate. Xiao et al. (1996) also develop a fixed-period AGA. A performance index, similar to the benefit-to-cost ratio of Tuson and Ross, measures the effectiveness of each operator and is used to rank the operators, with the highest ranked operator receiving the largest operator rate and the lowest ranked operator receiving the smallest operator rate. Chew et al. (2002) develop a variable-period AGA that employs two populations. The first population is a reference population with static operator rates while the operator rates in the second population are adjusted based on differences between the two populations.

Tabu search (TS), developed by Glover (1989, 1990a, 1990b), is a search heuristic that uses a single solution to search through a solution neighborhood, often utilizing both short-term and long-term memory. Park and Kim (1997) use a TS to schedule orders on identical parallel machines where each order has a ready time and a due date. An order is split into multiple jobs and each job is scheduled separately with the objective of minimizing the holding costs of the orders. Bilge et al. (2004) employ a TS to schedule uniform parallel machines with an objective of minimizing total tardiness. Jobs have sequence-dependent setups and unique arrival dates and due dates. Logendran and Subur (2004) develop a TS to solve $R\|\sum T_j$ with release times and machine availability times as wells as the possibility of splitting a job into two lots that can be processed on different machines. Since the success of a tabu search is based largely on the initial solution as well as the search procedure, they test four initial solution heuristics and six search procedures within the tabu search. Chen and Wu (2006) combine a tabu search with a threshold accepting method and an improvement method to solve $R\|\sum T_j$ with setups dependent upon the machine and the type of job it is processing.

The next section presents a brief overview of genetic algorithms and tabu search. Section 4 describes the three adaptive crossover selection schemes. Section 5 discusses the generation of test problems and the results of a computational study. Section 6 concludes and presents future work.

3. OVERVIEW OF GENETIC ALGORITHMS AND TABU SEARCH

Genetic algorithms were developed by Holland (1975) as a search and optimization heuristic that mimicked biological evolution and the Darwinian theory of survival of the fittest. A population of chromosomes is generated, where each chromosome represents a problem solution. Each chromosome is evaluated based on an evaluation function that determines the value of its solution. During subsequent generations, or iterations, chromosomes are randomly changed using genetic operators and then reevaluated. A reproduction scheme is used to change the chromosomes in the population in hopes of replacing some chromosomes currently in the population with new chromosomes with better evaluation function values. The population evolves with each new generation until a stopping criteria is met.

3.1 Chromosome structure

The chromosome structure for scheduling problems is unique in that each gene, or position, in the chromosome represents a job. Therefore, a chromosome will have N genes. A common representation is to randomly assign a number between 1 and M to each gene in the chromosome where this number represents the machine the job is assigned to. For instance, in a problem with 5 jobs and 2 machines, the chromosome [1, 2, 2, 1, 2] assigns jobs 1 and 4 to machine 1 and jobs 2, 3, and 5 to machine 2. In this representation, the order of jobs on each machine must still be found. Another common representation is to assign a job and a machine to each gene and then use the order of the genes to determine the machine sequences. For example, the chromosome [3-2, 1-1, 5-2, 2-2, 4-1] assigns jobs 3, 5, and 2 to machine 2 and jobs 1 and 4 to machine1 where they are processed in the given order.

A less common, but very effective representation is the random keys chromosome structure developed by Bean (1994). In this representation a random number from the distribution $U[1.00, M + 1)$ is generated for each gene. This number represents the machine the job is assigned to as well as the job's processing order. The integer part of the gene is the machine assignment. The decimal part of the gene is used to determine machine sequences. Jobs are first sorted by the integer part of the gene where jobs with the same integer are assigned to the machine represented by that integer. Within each machine, jobs are sorted based on the decimal part of the gene, where jobs with a smaller decimal value are processed before jobs with a larger decimal value. For example, the chromosome [1.21, 2.98, 2.10, 1.22, 2.25] assigns jobs 1 and 4 to machine 1 and jobs 3, 5, and 2 to machine 2 where they are processed in the given order. Since a random keys chromosome is just a sequence of random numbers, mutations or crossovers will not cause infeasible solutions. A crossover operation on a chromosome that assigns both the machine and job for each gene could result in a job being assigned to more than one gene and another job not being assigned at all, causing an infeasible solution. Using a chromosome that can become infeasible under crossovers requires an additional step to determine the feasibility of each chromosome, which is not needed for random keys genetic algorithms.

3.2 Genetic operators

Three genetic operators, elitist reproduction, crossover and immigration, are used to introduce randomness into the GA and hopefully prevent it from getting stuck in a local minimum. Elitist reproduction chooses the chromosomes with the best fitness (or evaluation function) value to remain in the population for the next generation. This keeps the strongest known solutions in the population. Immigration (Norman and Bean, 1999) randomly generates chromosomes to be added to the population for the next generation. Crossover randomly selects two "parent" chromosomes from the current population to create a "child" whose genes are a combination of its parents' genes.

During a single-point crossover (SPC), an integer n between 1 and N is selected as the crossover gene. For genes 1, ..., $n - 1$, parent 1's gene values are copied to child 1 and parent 2's gene values are copied to child 2. For genes n, ..., N, parent 1's gene values are copied to child 2 and parent 2's gene values are copied to child 1. During a two-point crossover (TPC), two integers n_1 and n_2 between 1 and N are selected as crossover genes where $n_1 < n_2$. For genes 1, ..., $n_1 - 1$, parent 1's gene values are copied to child 1 and parent 2's gene values are copied to child 2. For genes n_1, ..., $n_2 - 1$, parent 1's gene values are copied to child 2 and parent 2's gene values are copied to child 1. For genes n_2, ..., N, parent 1's gene values are copied to child 1 and parent 2's gene values are copied to child 2. During parametric uniform crossover (PUC), for each gene i = 1, ..., N, a random value is generated from $U(0,1)$. If this value is less than (W.L.O.G.) the probability of crossover P_c, then child 1 receives gene i from parent 1 and child 2 receives gene i from parent 2. If this value is greater than or equal to P_c, then child 1 receives gene i from parent 2 and child 2 receives gene i from parent 1. If the probability of a crossover P_c is limited to 0.5, the crossover is considered a uniform crossover (UC). Figure 1 shows an example of single-point, two-point, uniform, and parametric uniform crossovers.

3.3 Evaluation function, stopping criteria, and parameters

Our evaluation function is total weighted tardiness which we want to minimize. The GA will stop once a set number of iterations have passed since a better (lower) evaluation function value was found. The minimum number of iterations that must pass without finding a lower evaluation function value *MinIter* was set to 200. The traditional GA was tested with populations of size 100, 200, 300, 400, and 500. This initial testing showed that while a population size of 200 was statistically better than a population size of 100, there was not statistical difference between a population of size 200, 300, 400, and 500. Therefore, the number of chromosomes in the population *PopSize* was set to 200. For each generation, 20% of the population is generated through elitist reproduction, 79% is generated through crossover, and 1% is generated through immigration as suggested by Norman and Bean (1999).

3.4 Tabu search

To generate the initial solution for our tabu search, we will randomly generate *PopSize* solutions, where *PopSize* is the number of chromosomes in our GA's population, and evaluate these solutions based on the same objective function used to judge chromosome fitness in the GA, total tardiness. During a job swap, two jobs switch positions in their respective schedules. These jobs may both be scheduled on the same machine or they may be scheduled on different machines. During a job insert, a job is randomly placed in a new position in the schedule of either its current machine or the schedule of another machine. All possible job swaps and job inserts will be performed on this solution with the result of each neighborhood move added to the candidate list. This results in $(N(N-1))/2$ swaps and $N(M-1)$ inserts. After all possible neighborhood moves have been made, the candidate list is evaluated and the best solution from the candidate list is checked to determine if it is tabu. The tabu list will hold the objective value of the seven most recent starting solutions. A tabu list length of seven has been shown to be effective in machine scheduling (Glover, 1990a; Glass et al., 1994; Hsieh et al., 2003). If the selection solution is not tabu, then it becomes the starting solution for the next iteration. If the selected solution is tabu, it is then checked against the aspiration criteria. If the solution has a better objective value than the best known solution, then it will become non-tabu and will be chosen as the starting solution for the next iteration. Otherwise it remains tabu and the next best solution is checked to see if it is tabu. This continues until a tabu solution exceeds the aspiration criteria or a non-tabu solution is found. If all solutions are tabu, the best solution in the candidate list is selected. After choosing the next starting solution, its objective value is added to the tabu list and the objective value that has been in the tabu list the longest is removed from the tabu list. The TS employs the same stopping criteria as the GA: 200 iterations without a new best solution.

Parent 1	1.23	3.42	2.99	3.23	1.22	2.00
Parent 2	3.45	2.35	1.67	2.84	1.11	3.75
Child under SPC $n = 3$	1.23	3.42	1.67	2.84	1.11	3.75
Child under TPC $n_1 = 2, n_2 = 4$	1.23	2.35	1.67	3.23	1.22	2.00
Child under UC	1.23	2.35	2.99	2.84	1.22	3.75
Crossover Prob	0.25	0.65	0.37	0.99	0.01	0.86
Child under PUC	1.23	3.42	2.99	2.84	1.22	3.75
Crossover Prob	0.25	0.65	0.37	0.99	0.01	0.86

Figure 1. Example of SPC, TPC, UC, and PUC.

Parent 1	Solution Genes	Reg Gene
Parent 2	Solution Genes	Reg Gene

		Parent 2	
		0	**1**
Parent 1	**0**	00	01
	1	10	11

00	Single-Point Crossover
01	Two-Point Crossover
10	Uniform Crossover
11	Parametric Uniform Crossover

Figure 2. Regulatory gene pairings for the 0-1 RGS representation.

		Parent 2		
		0	**1**	**2**
Parent 1	**0**	00	01	02
	1	10	11	12
	2	20	21	22

02	10	21	Two-Point Crossover
01	11	20	Uniform Crossover
00	12	22	Parametric Uniform Crossover

Figure 3. Regulatory gene pairings for the 0-1-2 RGS representation.

4. ADAPTIVE CROSSOVER SELECTION SCHEMES

Three adaptive crossover selection schemes (regulatory gene selection, roulette wheel selection, and tournament selection) will be employed during the crossover phase of a GA. Single-point crossover (SPC), two-point crossover (TPC), uniform crossover (UC), and parametric uniform crossover (PUC) will be tested under the adaptive crossover selection schemes while the traditional GA will use only PUC.

4.1 Regulatory gene selection

While biological cell reproduction occurs within the nucleus of the cell, outside genetic material sometimes controls the reproduction process. We will mimic this phenomenon in our GA by adding a regulatory gene to each chromosome. Whenever a crossover operation is required, two parents are randomly chosen and their regulatory genes are combined to select the crossover operator to be performed. The first regulatory gene representation is the 0-1 representation. In this representation, each regulatory gene receives a value of 0 or 1. Figure 2 shows the regulatory gene pairings for the 0-1 representation. In this representation, each crossover operator is represented by one regulatory gene pairing. Initial testing shows that during the use of this representation, either the value of 0 or 1 quickly dominates the other value, making it almost impossible to perform three of the crossover operator. For example, if most of the chromosomes in the population contain a regulatory gene value of 0, the crossover operators TPC, UC, and PUC will rarely occur.

The 0-1-2 representation was developed to prevent the GA from converging to a single regulatory gene value and therefore, a single crossover operator. In this representation, each regulatory gene receives a value of 0, 1, or 2. Figure 3 shows the regulatory gene pairings for this representation. This representation does not include SPC because it is a subset of TPC. With this representation, if one of the regulatory gene values is dominated by the other values, each crossover operator can still be selected and the GA is less likely to converge to a single crossover operator.

The Rand representation was also developed to prevent the GA from converging to a single regulatory gene value. The regulatory gene receives a random value from $U(0,1)$. When a crossover operation is required, two random values r_1 and r_2 are generated from $U(0,1)$. The regulatory gene value of parent 1 is compared to r_1 and the regulatory gene value of parent 2 is compared to r_2. The results of these comparisons determine the crossover operator to be performed. Since r_1 and r_2 are randomly generated each time a crossover operation is required, none of the crossover operators will be able to dominate the other crossover operators.

The Species representation was developed to mimic natural environments where several species coexist. These species are similar enough to compete for the same resources, but are different enough to not reproduce with other species. In this representation, each crossover operator will be considered a separate species, which will be denoted by an additional gene in chromosome. The starting population will be split evenly among the different species. When a crossover is required, two parents will be randomly chosen until the parents contain the same species gene. The crossover operator designated by this species gene will then be performed on these parents. Elite reproduction and immigration will be performed without regard to the species gene.

4.2 Roulette wheel selection

Roulette wheel selection (Goldberg, 1989) is often used in reproduction because it rewards chromosomes based on their fitness in proportion to the overall fitness of the entire population. This encourages the evolutionary procedure to focus on the strongest portion of the population while occasionally allowing the inclusion of a weaker chromosome. We will use roulette wheel selection (RWS) to choose a crossover operator by awarding "credits" to a crossover operator for producing "good" children. Each crossover operator will receive the same base roulette value *b*. Then, whenever a crossover operator produces a child that has a lower total tardiness value than the current lowest total tardiness value, its roulette value will increase by one.

The use of a roulette wheel allows the GA to choose crossover operators based on their past performance. Operators that produce better children will have higher roulette values and should therefore be chosen more often than operators that produce worse children and thus have lower roulette values. Since an operator's roulette value can increase throughout the run of the GA, the initial worth of each operator does not determine the outcome of the entire run. Operators who produce poor children in the early stages of the GA, but then produce many good children in the later stages will gradually increase their portion of the roulette wheel to reflect their growing merit to the GA. Conversely, operators who produce good children in the early stages, and thus initially comprise a large portion of the roulette wheel, can later lose much of their share of the roulette wheel if they fail to continue to produce good children.

The choice of a base value can significantly affect the overall performance of this representation. For the first iteration, every crossover operator has a 25% chance of being chosen whenever a crossover operation is required. If a base value of 1 is used, then the first crossover operator that produces a "good" child will give that crossover operator a 40% chance of being chosen, an increase of 160%. But if a base value of 100 is used, the first crossover operator that produces a "good" child will give that crossover operator a 25.19% chance of being chosen, only a 101% increase. Since the base value determines how much a single "good" child affects the overall distribution of the roulette wheel among each crossover operator, we will test base values of 1, 50, 100, and 150.

4.3 Tournament selection

Tournament selection is often used within a crossover operation to choose the best child among all the children created by this crossover. This will ensure that the best child produced by the crossover will always continue to the next generation. Whenever a crossover operation is required, two parents will be chosen and each of the four crossover operator will be performed on these parents. Tournament selection (TMT) will then be applied to this pool of children and the best child will continue to the next generation.

5. COMPUTATIONAL EVALUATION AND RESULTS

The adaptive crossover selection genetic algorithms (ACSGAs) and TS were tested against a traditional GA on a Pentium IV 3.8 GHz with 1 GB of RAM. All programs were coded in C and run in Visual Studio .NET with Windows XP as the operating system. Processing times were generated from the distribution $U[100, 200]$. Due dates are dependent upon processing times and were generated from two distributions generated by the formula $U[P(1-\tau-\rho/2), P(1-\tau+\rho/2)]$, where P controls the makespan, τ controls the priority factor, and ρ controls the due date range where $\rho = 0.5$ and $\tau = 0.3$ (L) or 0.6 (H), as suggested by Kim et al. (2003). The due date formula generates a tight (low variability due to $\tau = 0.3$) due date distribution and a loose (high variability due to $\tau = 0.6$) due date distribution. P is estimated as the average processing time of a job multiplied by the average number of jobs per machine (the number of jobs divided by the number of machines). The setup times are asymmetric (i.e. s_{ijk} may not be equal to s_{jik}) and are generated from the distributions $U[40,60]$ (L), $U[20,80]$ (M), and $U[0,100]$ (H), where 50 is the mean setup time. This allows for setups with low, medium, and high variability. The setup times satisfy the triangle inequality (i.e. $s_{ijk} + s_{jlk} \geq s_{ilk}$). The following problem sizes were considered: 50 jobs with 5 or 10 machines, 100 jobs with 5, 10, or 20 machines, and 200 jobs with 10 or 20 machines. Ten problem instances were generated for each of the 6 problem sets (setup distribution, and τ value) within each problem size (i.e. each job-machine combination). Each algorithm was run sixteen times for each problem instance. Table 1 shows the average total tardiness of the GA, ACSGAs, and TS for each problem size.

The GA, ACSGAs, and TS will be compared based on solution quality using Fisher's LSD test, which is basically a multiple t-test. When an algorithm is statistically worse than the GA for all six problem sets for two consecutive iterations (problem sizes), this algorithm will be excluded from future comparisons. By requiring two consecutive iterations with inferior results, we ensure that the poor performance of the specific algorithm is not due to randomness. Table 2 shows the LSD comparison of average tardiness of the GA, ACSGAs, and TS for several problem sizes. A ✕ denotes that the algorithm is statistically worse than the GA (i.e. the algorithm and the GA are statistically different and the algorithm has a higher average total tardiness than the GA) while a ✓ denotes that the algorithm is statistically better than the GA (i.e. the algorithm and the GA are statistically different and the algorithm has a lower average total tardiness than the GA). Otherwise, the GA and algorithm are statistically the same.

In the first iteration (50 jobs and 5 machines), RGS with the Species representation and TS were statistically worse than the GA for all six problem sets while RWC with b = 50, 100, and 150 were statistically worse than the GA for at least one problem set. No ACSGA schemes were statistically better than the GA. In the second iteration (50 jobs and 10 machines), RGS with the Species representation, RWS with b = 150, and TS were all statistically worse than the GA for all six problem sets while RWS with b = 50 and RWS with b = 100 were statistically worse than the GA for at least one problem set. This is the second consecutive iteration that RGS with the Species representation and TS were statistically worse than the GA for all problem sets. Therefore, the ACSGA scheme and TS are inferior to the GA. No ACSGA schemes were statistically better than the GA.

In the third iteration (100 jobs and 5 machines), RGS with the Species representation and TS are not compared against the GA due to their previous performances. RGS with the Rand representation and RWS with b = 1 were statistically worse than the GA for all six problem sets. TMT was statistically better than the GA for three of the six problem sets. Since RWS b = 150 was not statistically worse than the GA for all problem sets (it was actually statistically the same for all six problem sets), it will not be excluded from further iterations.

In the fourth iteration (100 jobs and 10 machines), RGS with the Rand representation and RWS with b = 1 were statistically worse than the GA for all six problem sets while RWS with b = 50, 100, and 150 were statistically worse than the GA for at least one problem set. This is the second consecutive iteration the RGS with the Rand representation and RWS with b = 1 were statistically worse than the GA for all problem sets. Therefore, these ACSGA schemes are inferior to the GA. For the second consecutive iteration, TMT was statistically better than the GA for at least one problem set. In this iteration, it was statistically better for five of the six problem sets.

Table 1. Average total tardiness for the GA, ACSGAs, and TS

N-M	GA	RGS 0-1	RGS 0-1-2	RGS Rand	RGS Species	RWS b = 1	RWS b = 50	RWS b = 100	RWS b = 150	TMT	TS
50-5	9994	10037	9992	10157	12466	10185	10310	10310	10312	9871	10576
50-10	5966	6019	5961	6107	7442	6117	6208	6211	6222	5863	6833
100-5	40602	40764	40573	41797	-	41915	41152	41101	41034	39705	-
100-10	23086	23283	23059	23889	-	23916	23445	23432	23383	22676	-
100-20	14192	14396	14183	-	-	-	14427	14356	14377	13978	-
200-10	96338	97185	96052	-	-	-	99293	98981	98956	91196	-
200-20	53896	54603	53825	-	-	-	55793	55488	55547	51446	-

Table 2. LSD average tardiness comparison for the GA, ACSGAs, and TS

N *M*	Setup Value	RGS 0-1		RGS 0-1-2		RGS Rand		RGS Species		RWS b = 1		RWS b = 50		RWS b = 100		RWS b = 150		TMT		TS	
	τ Value	L	H	L	H	L	H	L	H	L	H	L	H	L	H	L	H	L	H	L	H
50 5	L							×	×				×				×			×	×
	M							×	×			×		×		×	×			×	×
	H							×	×			×		×		×				×	×
50 10	L							×	×			×	×	×	×	×	×			×	×
	M							×	×			×	×	×	×	×	×			×	×
	H							×	×			×		×		×	×			×	×
100 5	L					×	×			×	×							✓			
	M					×	×			×	×							✓	✓		
	H					×	×			×	×										
100 10	L					×	×			×	×		×		×		×		✓		
	M					×	×			×	×			×			×	✓	✓		
	H					×	×			×	×	×		×				✓	✓		
100 20	L											×	×								
	M																				
	H																				
200 10	L											×	×	×	×	×	×	✓	✓		
	M											×	×	×	×	×	×	✓	✓		
	H	×										×	×	×	×	×	×	✓	✓		
200 20	L	×										×	×	×	×	×	×	✓	✓		
	M		×									×	×	×	×	×	×	✓	✓		
	H											×	×	×	×	×	×	✓	✓		

Legend:
× Statistically worse than the GA
✓ Statistically better than the GA

Table 3. Average solving time (seconds) for the GA, ACSGAs, and TS

N-M	GA	RGS 0-1	RGS 0-1-2	RGS Rand	RGS Species	RWS b = 1	RWS b = 50	RWS b = 100	RWS b = 150	TMT	TS
50-5	3.38	3.61	3.57	4.18	3.42	3.90	2.11	2.15	2.19	9.16	8.62
50-10	3.21	3.51	3.45	3.61	3.24	3.72	2.07	2.05	2.01	9.05	11.00
100-5	8.36	9.21	8.48	9.16	-	10.38	8.51	8.36	8.18	41.36	-
100-10	9.91	11.01	10.07	10.31	-	11.43	9.88	9.79	9.74	40.81	-
100-20	11.16	11.20	12.29	-	-	-	11.12	11.28	11.13	38.77	-
200-10	12.11	13.08	13.87	-	-	-	11.96	11.90	11.84	48.54	-
200-20	12.97	14.50	14.93	-	-	-	12.30	12.22	12.06	52.72	-

Table 4. Maximum Solving Time (seconds) for the GA, ACSGAs, and TS

N-M	GA	RGS 0-1	RGS 0-1-2	RGS Rand	RGS Species	RWS $b = 1$	RWS $b = 50$	RWS $b = 100$	RWS $b = 150$	TMT	TS
50-5	4.12	4.36	4.20	6.01	4.98	4.54	2.59	2.73	2.69	11.04	10.78
50-10	3.71	4.21	4.15	4.25	4.51	4.55	2.46	2.57	2.38	10.90	14.40
100-5	11.08	11.22	10.46	11.14	-	12.26	11.12	10.55	10.93	48.27	-
100-10	12.24	12.97	11.70	12.41	-	14.06	11.60	10.98	11.10	53.29	-
100-20	12.36	12.99	14.31	-	-	-	12.97	13.07	12.56	46.69	-
200-10	13.78	15.04	16.14	-	-	-	14.35	15.19	13.76	59.08	-
200-20	14.60	16.61	17.81	-	-	-	14.01	13.53	13.52	63.48	-

In the fifth iteration (100 jobs and 20 machines), RGS with the Rand representation and RWS with $b = 1$ are not compared with the GA due to their previous poor performance. RWS $b = 50$ was statistically worse than the GA for at least one problem set, but there were no ACSGA schemes that were statistically worse than the GA for all six problem sets. Also, no ACSGA schemes were statistically better than the GA.

In the sixth iteration (200 jobs and 10 machines), RWS with $b = 50$, 100, and 150 were all statistically worse than the GA for all six problem sets while TMT was statistically better than GA for all six problem sets. In the seventh iteration (200 jobs and 20 machines), RWS with $b = 50$, 100, and 150 were all statistically worse than the GA for all six problem sets. This is the second consecutive iteration these ACSGA schemes were statistically worse than the GA for all problem sets, therefore, these schemes are inferior to the GA. Also, for the second consecutive iteration, TMT was statistically better than the GA for all six problem sets. Therefore, we can conclude that TMT performs better than the GA, RGS with the 0-1 and 0-1-2 representations perform equally to the GA, and TS, RGS with the Rand and Species representations, and RWS with all levels of b perform worse than the GA. Table 3 shows the average solving time (in seconds) of the GA, ACSGAs, and TS while Table 4 shows the maximum solving time (in seconds). The average and maximum solving times are similar for the GA and all of the ACSGAs except TMT and TS. The average solving time of TMT is two to five times longer than that of the GA. This increase in solving time of the TMT is mainly due to its crossover selection procedure. Each time a crossover operation is needed in TMT, four crossover operations are performed with the best child proceeding to the next generation. Therefore, TMT is generating four times more crossovers per iteration than the GA and other ACSGAs.

6. CONCLUSIONS AND FUTURE WORK

Scheduling unrelated parallel machines to minimize completion time without any other considerations such as setups or due dates is an *NP*-hard problem. Therefore, adding due dates and sequence-dependent setups with an objective to minimize total tardiness makes the problem difficult to solve for even small problem sizes and nearly impossible to solve with most traditional solution techniques for the larger sized problems studied in this paper.

Genetic algorithms (GAs) are often able to find near-optimal solutions to problems while maintaining performance as the problem size increases. We have attempted to improve the performance of a traditional GA by allowing it to choose from several crossover operators whenever a crossover operation is required. This allows the GA to adapt to the different conditions that arise during the run of the GA that might favor one crossover operator over another.

Three adaptive crossover selection genetic algorithms (ACSGAs) and a tabu search (TS) were tested against a traditional GA for solution quality over seven problem sizes ranging from 50 jobs and 5 machines to 200 jobs and 20 machines. The regulatory gene selection (RGS) scheme with Species representation and TS performed worse than the GA and other ACSGA schemes for even the smallest problem size of 50 jobs and 5 machines. The RGS with the Rand representation and the roulette wheel selection (RWS) scheme with a base value $b = 1$ were the next ACSGA schemes to be excluded from further iterations after 100 jobs and 10 machines. RWS with a base value of $b = 50$, 100, and 150 were excluded from further iterations after 200 jobs and 20 machines. RGS with the 0-1 and 0-1-2 representations were found to be statistically the same as the GA even for 200 jobs and 20 machines. The tournament selection (TMT) scheme was the only ACSGA scheme that was statistically better than the GA. For 200 jobs and 10 machines and 200 jobs and 20 machines problems, TMT outperformed the GA for all six problem levels.

While TMT produces solutions superior to the GA's solutions, it also requires two to five times as much solving time, on average. TMT performs four crossover operations each time a crossover is required so it is reasonable that it would require more solving time. As long as there is not a constraint on the amount of solving time, TMT produces better solutions. If there is a constraint on the amount of solving time that can be allowed, the traditional GA might be sufficient.

The effectiveness of RWS is heavily dependent upon the choice of the base credit value as well as the first few successful crossover operations. Because the first few successful crossovers have such a large effect on the proportioning of the roulette wheel, crossover operators that initially produce better children could gain an unfair advantage. These operators will be chosen more often due to their larger roulette values, and therefore have more opportunities to produce good children and continue to increase their roulette values. This could cause a cycle that would favor these operators even after they stop produce better children. To prevent this possible bias, as well as increase the overall performance of RWS, the roulette credits given for good children could increase as the run of the GA continued, rather than remaining static. This would allow good children produced late in the run of the GA, when it is harder to find better solutions, to have a greater effect on the roulette values than those good children produce early in the run of the GA, when it is relatively easy to find better solutions.

Future work will focus on forming serial batches of jobs with different job types and scheduling these batches on unrelated parallel machines. A new chromosome interpretation, based on the random keys representation used in this research, will be used to simultaneously batch and schedule jobs on these unrelated parallel machines. This new chromosome interpretation will be tested with the traditional GA, the RWS AGA, and the RGS AGA with the 0-1 and 0-1-2 representations as well as with TMT to determine if TMT will continue to outperform the GA and other ACSGAs under a more complex problem. The use of an increasing roulette credit with the RWS AGA will also be implemented to test its effect on the quality of solutions produced by the AGA.

REFERENCES

1. Alidaee, B. and Panwalkar, S.S. (1993). Single stage minimum absolute lateness problem with a common due date on non-identical Machines. *Journal of the Operational Research Society*, 44(1): 29-36.
2. Bank, J. and Werner, F. (2001). Heuristic algorithms for unrelated parallel machine scheduling with a common due date, release dates, and linear earliness and tardiness penalties. *Mathematical and Computer Modelling*, 33(4-5): 363-383.
3. Bean, J.C. (1994). Genetic algorithms and random keys for sequencing and optimization. *ORSA Journal on Computing*, 6(2): 154-160.
4. Bilge, U., Kirac, F., Kurtulan, M., and Pekgun, P. (2004). A tabu search algorithm for parallel machine total tardiness problem. *Computers and Operations Research*, 31: 397-414.
5. Chen, J.F. and Wu, T.H. (2006). Total Tardiness minimization on unrelated parallel machine scheduling with auxiliary equipment constraints. *OMEGA*, 34(1): 81-89.
6. Cheng, T.C.E., Chen, Z.L., and Li, C.L. (1996). Parallel-machine scheduling with controllable processing times. *Institute of Industrial Engineers Transactions*, 28(2): 177-180.
7. Cheng, R., Gen, R., and Tosawa, T. (1995). Minmax earliness/tardiness scheduling in identical parallel machines system using genetic algorithms. *Computers and Industrial Engineering*, 29(1-4): 513-517.
8. Chew, E.P., Ong, C.J., and Lim, K.H. (2002). Variable period adaptive genetic algorithm. *Computers and Industrial Engineering*, 42: 353-360.
9. Cochran, J.K., Horng, S.M., and Fowler, J.W. (2003). A multi-population genetic algorithm to solve multi-objective scheduling problems for parallel machines. *Computers and Operations Research*, 20: 1087-1102.
10. Davis, L. (1989). Adapting operator probabilities in genetic algorithms. In: J.D. Schaffer (Ed.), *Proceedings of the Third International Conference on Genetic Algorithms and their Applications*, San Mateo, pp. 61-69.
11. Glass, C.A., Potts, C.N., and Shade, P. (1994). Unrelated parallel machine scheduling using local search. *Mathematical and Computer Modelling*, 20(2): 41-52.
12. Glover, F. (1989). Tabu search−Part I. *ORSA Journal on Computing*, 1(3): 190-206.
13. Glover, F. (1990a). Tabu search: A tutorial. *Interfaces*, 20(4): 74-94.
14. Glover, F. (1990b). Tabu search−Part II. *ORSA Journal on Computing*, 2(1):4-32.
15. Goldberg, D.E. (1989). *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley, Reading, Massachusetts.
16. Graham, R.L., Lawler, E.L., Lenstra, J.K., and Rinnooy Kan, A.H.G. (1979). Optimization and approximation in deterministic sequencing and scheduling: A survey. *Annals of Discrete Mathematics*, 5: 287-326.
17. Holland, J.H. (1975). *Adaptation in Natural and Artificial Systems*, The University of Michigan Press, Ann Arbor.
18. Hsieh, J.C., Chang, P.C., and Hsu, L.C. (2003). Scheduling of drilling operations in printed circuit board factory. *Computers and Industrial Engineering*, 22: 461-473.
19. Jou, C. (2005). A genetic algorithm with sub-indexed partitioning genes and its application to production scheduling of parallel machines. *Computers and Industrial Engineering*, 48: 39-54.
20. Julstrom, B.A. (1995). What have you done for me lately? Adapting operator probabilities in a steady-state genetic algorithm. *Proceedings of the Sixth International Conference on Genetic Algorithms*, University of Pittsburgh, pp. 81-87.
21. Kim, D.W., Kim, K.H., Jang, W., and Chen, F.F. (2002). Unrelated parallel machine scheduling with setup times using simulated annealing. *Robotics and Computer Integrated Manufacturing*, 18(3-4): 223-231.
22. Kim, D.W., Na, D.G., and Chen, F.F. (2003). Unrelated parallel machine scheduling with setup times and a total weighted tardiness objective. *Robotics and Computer Integrated Manufacturing*, 19(1-2):173-181.
23. Kurz, M.E. and Askin, R.G. (2001). Heuristic scheduling of parallel machines with sequence-dependent set-up times. *International Journal of Production Economics*, 39(16): 3747-3769.
24. Kurz, M.E. and Askin, R.G. (2004). Scheduling flexible flow lines with sequence-dependent setup times. *European Journal of Operational Research*, 159: 66-82.
25. Liaw, C.H., Lin, Y.K., Cheng, C.Y., and Chen, M. (2003). Scheduling unrelated parallel machines to minimize total weighted tardiness. *Computers and Operations Research*, 30(12): 1777-1789.

26. Logendran, B. and Subur, F. (2004). Unrelated parallel machine scheduling with job splitting. *Institute of Industrial Engineers Transactions*, 36(4): 359-372.
27. Luu, D.T., Bohez, E.L.J., and Techanitisawad, A. (2002). A hybrid genetic algorithm for the batch sequencing problem on identical parallel machines. *Production Planning and Control*, 13(3): 243-252.
28. Min, L. and Cheng, W. (1999). A genetic algorithm for minimizing the makespan in the case of scheduling identical parallel machines. *Artificial Intelligence in Engineering*, 13: 399-403.
29. Norman, B.A. and Bean, J.C. (1999). A genetic algorithm methodology for complex scheduling problems. *Naval Research Logistics*, 46: 199-211.
30. Park, M.-W. and Kim, Y.-D. (1997). Search heuristics for a parallel machine scheduling problem with ready times and due dates. *Computers and Industrial Engineering*, 33(3-4): 793-796.
31. Randall, P. and Kurz, M.E. (2005). Scheduling unrelated parallel machines using a random keys genetic algorithm. *Proceedings of the IIE 2005 Annual Conference*, Atlanta GA.
32. Sivrikaya-Serifoglu, F. and Ulusoy, G. (1999). Parallel machine scheduling with earliness and tardiness penalties. *Computers and Operations Research*, 26: 773-787.
33. Tuson, A. and Ross, P. (1996). Cost based operator rate adaptation: An investigation. In: H.M. Voigt, W. Ebeling, I. Rechenberg, and H.P. Schwefel (Eds.), *Parallel Problem Solving from Nature-PPSN IV*, Springer, Berlin.
34. Van Hop, N. and Nagarur, N.N. (2004). The scheduling problem of PCBs for multiple non-identical parallel machines. *European Journal of Operational Research*, 158: 577-594.
35. Van Hop N. and Tabucanon, M.T. (2005). Adaptive genetic algorithm for lot-size problem with self-adjustment operation rate. *International Journal of Production Economics*, 98(2): 129-135.
36. Xiao, J., Michalewicz, Z., and Zhang, L. (1996). Evolutionary planner/navigator operator performance and self-tuning. *Proceedings of IEEE International Conference on Evolutionary Computation*, pp. 366-371.