IEMS Vol. 10, No. 1, pp. 15-23, March 2011.

Heuristic Algorithms for Parallel Machine Scheduling Problems with Dividable Jobs

Chi-Yang Tsai†
Department of Industrial Engineering and Management, Yuan Ze University, Taoyuan, Taiwan
Tel: +886-3-463-8800, E-mail: iecytsai@satyrun.yzu.edu.tw

You-Ren Chen
Department of Industrial Engineering and Management
Yuan Ze University, Taoyuan, Taiwan

Received, January 18, 2011; Revised, February 19, 2011; Accepted, February 21, 2011

Abstract. This research considers scheduling problems with jobs which can be divided into sub-jobs and do not required to be processed immediately following one another. Heuristic algorithms considering how to divide jobs are proposed in an attempt to find near-optimal solutions within reasonable run time. The algorithms contain two phases which are executed recursively. Phase 1 of the algorithm determines how jobs should be divided while phase 2 solves the scheduling problem given the sub-jobs established in phase 1. Simulated annealing and genetic algorithms are applied for the two phases and four heuristic algorithms are established. Numerical experiment is conducted to determine the best parameter values for the heuristic algorithms. Examples with different sizes and levels of complexity are generated. Performance of the proposed algorithms is evaluated. It is shown that the proposed algorithms are able to efficiently and effectively solve the considered problems.

Keywords: Parallel Machine Scheduling, Dividable Jobs, Simulated Annealing, Genetic Algorithm

1. INTRODUCTION

Parallel machine scheduling problems are commonly seen in practice. A group of jobs are to be assigned to a group of machines and the processing sequences of jobs on each machine are determined as well. In many cases, a job contains several stages that are to be processed in sequence. Such a job can be divided into several smaller sub-jobs with each stage representing a sub-job. Processing of sub-jobs does not have to be in an immediate sequence. They do not even have to be processed on the same machine. Examples of dividable jobs can be found in automotive part processing. The manufacturing process of certain automotive parts includes spray painting. Different parts require various numbers of layers of paint. The layers of paint on one part have to be sprayed on in a specific sequence but the operations do not have to be done with one immediately following another. Therefore, one can work on a part by spraying on all the necessary layers of paint and then move on to another part. Or one can work on a particular layer of paint (a particular color, for example) to all the parts that require it, and then switch to work on another layer of paint.

Apparently, allowing job splitting provides more options in setting up job schedules and possibly finding better solutions. However, dividing jobs into sub-jobs implies increasing numbers of jobs (sub-jobs) to be scheduled and needs greater effort to find better job schedules. Consequently, splitting jobs into the most number of sub-jobs allowed is not necessary the best option due to long search time for finding better solutions. Questions arise that how to determine proper ways to divide jobs to improve solution quality, and maintain search effort for better solutions within acceptable range at the same time. The goal of this study is to develop heuristic algorithms by applying simulated annealing and genetic algorithms for effectively and efficiently solving the parallel machine scheduling problem with dividable jobs to minimize total tardiness.

There are many studies on parallel machine scheduling problems, including Azizoglu and Kirca (1998), Dessouky *et al*. (1998), Azizoglu and Kirca (1999), Yalaoui and Chu (2002), Bilge *et al*. (2004), and Armentano and de França Filho (2007). Tan *et al*. (2000) considered sequence-dependent setup time and evaluated the performance of several technique, including branch-and-bound method, genetic algorithm and simulated annealing al-

† : Corresponding Author

gorithm. Liao and Lin (2003) studied makespan minimization for two uniform parallel machines and compared two methods, LPT and MULTIFIT.

Several studies utilized simulated annealing algorithm on parallel machine scheduling problems, including Piersma and Dijk (1996), Anagnostopoulos and Rabadi (2002), Kim *et al*. (2002), Kim *et al*. (2003), and Low (2005). Kim *et al*. (2006) approached unrelated parallel machine scheduling problems with sequence-dependent setup time using a number of heuristic method, including simulated annealing algorithm. Guo *et al*. (2007) applied simulated annealing and tabu search to unrelated parallel machine scheduling problems with an objective of minimizing makespan. Genetic algorithm is another common heuristic algorithm applied to machine scheduling problems, such as Mason (1992), Cheng *et al*. (1995), Rubin and Ragatz (1995), Min and Cheng (1999) and Tan *et al*. (2000).

The rest of the paper is organized as follows. The next section describes the settings on the considered dividable jobs and scheduling problems. Section 3 presents the proposed two-phase heuristic algorithm and how it is designed utilizing simulated annealing algorithm and genetic algorithm. Section 4 introduces an index to evaluate the complexity level of test instances and generation of test instances. The conducted numerical experiment and analytical results are presented in Section 5. This paper concludes in Section 6.

2. PROBLEM DECRIPTION

This paper considers the problem of scheduling dividable jobs on identical parallel machine with an objective to minimize total tardiness. It is assumed that jobs can be divided into sub-jobs. Sub-jobs divided from the same job have to be processed in a specific sequence but not necessary immediately follow one another. Taking a three-stage job as an example, the process of job A contains three stages, A_1, A_2 and A_3, or equivalently, there are two cut points. Stage A_1 has to be finished before stage A_2 can be processed. Likewise, stage A_3 can only be processed after stage A_2 is completed. However, the processing of stage A_2 does not have to begin right after the completion of stage A_1. Similarly, stage A_3 can be assigned to another machine after stage A_2 is finished.

Figure 1 shows the four possible settings of processing job A. The three stages can be processed in sequence with one stage immediately following another, as illustrated by setting 1. They can also be processed separately, even on different machines, as shown by setting 2. In this case, job A is divided into three sub-jobs, each containing one stage. In the third setting, stage A_1 and stage A_2 form a sub-job and are processed in immediate sequence. Stage A_3 is treated as a sub-job and can be processed later. In the last setting, there are also two sub-jobs. Stage A_2 and stage A_3 form a sub-job which can be processed after the sub-job consisting of stage A_1 is done. Sub-jobs from the same job cannot be processed at the same time.

Splitting jobs into smaller sub-jobs increases flexibility of job scheduling, thus better job schedules can be found. However, it also increases the number of jobs to be scheduled and more effort is needed to search for better solutions. For example, a group of 10 jobs are to be scheduled for process. Each job contains 3 stages and can be divided in the way as illustrated in Figure 1. If every job is divided into 3 sub-jobs, the effective number of jobs to be scheduled increases from 10 to 30. Effort on searching for optimal or near-optimal solutions rises exponentially as the number of jobs increases. Granted, chances to find the optimal solution are the highest if jobs are divided into as many sub-jobs as possible. However, it is impractical if there too many jobs to be scheduled.

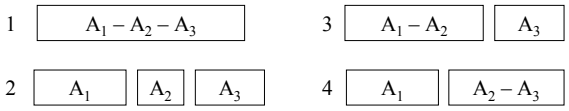

Figure 1. Possible settings of processing a 3-stage job.

The aim of this study is to utilize the characteristic of dividable jobs and develop methods to properly split jobs to increase scheduling flexibility, leading to better solutions, and control the search effort within reasonable range at the same time. For this purpose, heuristic algorithms are developed to find a balance between solution quality and search efficiency.

The following assumptions are made in this study.

1. The numbers of jobs and machines are known and fixed.
2. Available cut points of each job are known and fixed.
3. The processing time of each stage of a job is known.
4. The processing time of a sub-job is the sum of the processing times of the job stages in the sub-job.
5. A machine can only process one job (sub-job) at a time.
6. Job (sub-job) processing is non-preemptive.
7. The due date of each job is known.
8. Sequence-dependent setup time is not considered.
9. A job is completed when the last stage of the job is finished.

3. ALGORITHMS

To utilize the property of dividable jobs, mixed heuristic algorithms consisting of two phases are developed. The first phase determines how jobs are divided and a set of sub-jobs is established. A transition process converts this set of sub-jobs into the initial setting of the second phase which then searches for best sub-job as-

signments and sequences on the given set of identical parallel machines. The objective value of the schedule obtained in phase 2 is fed back to phase 1 as a reference for modifying how jobs are to be divided. The two phases are executed iteratively until termination conditions are met. Figure 2 illustrates the procedure of the proposed heuristic algorithms.

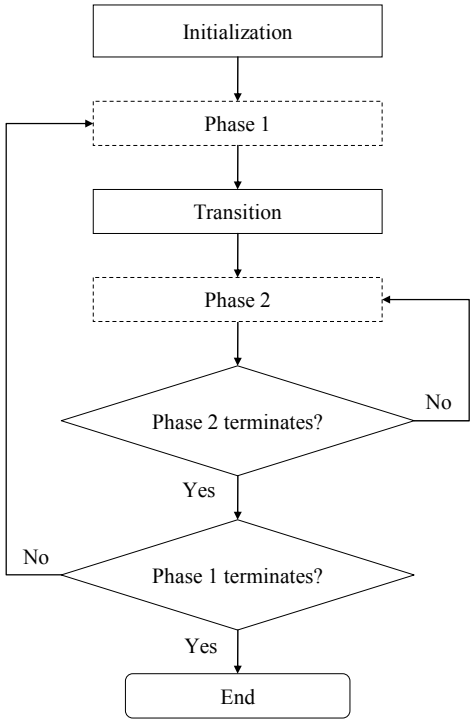

Figure 2. Flow chart of the two-phase algorithm.

Simulated annealing algorithm (SA) and genetic algorithm (GA) are applied in the development of the heuristic algorithms. Both SA and GA can be utilized for the two phases of the algorithm. As a result, there are four possible combinations. For example, the algorithm denoted as SA-SA means both phase 1 and phase 2 of the algorithm are developed using SA. In the initialization step, an initial state for phase 1 of the algorithm (designed based on SA) representing a set of sub-jobs established from the group of jobs to be scheduled is generated and an assignment and sequences of this set of sub-jobs are determined based on some simple decision rules. The resulting total tardiness is assigned as the fitness value of this state. Next, phase 1 generates a new state, S^{new} (a new set of sub-jobs). Through transition process, an initial state (assignment and sequences of S^{new}) is established for phase 2 algorithm (developed based on SA). The phase 2 algorithm then search for optimal assignment and sequences of S^{new}. After phase 2, algorithm stops according to a pre-determined termination condition. Total tardiness of the obtained optimal (or sub-optimal) solution is assigned as the fitness value of S^{new}. Phase 1 algorithm's termination conditions are then examined. If conditions are met, the whole heuristic algorithm stops. Otherwise, go to phase 1 algorithm and generate a new state.

The algorithm with both phases designed using GA is denoted as GA-GA. In algorithm SA-GA, SA is applied to phase 1 and GA is applied to phase 2. Finally, phase 1 algorithm is designed using GA and phase 2 is designed using SA in algorithm GA-SA.

The pseudo code of the GA-SA algorithm is shown in Figure 3. The detailed design of the GA and SA algorithms can be referred to Tsai and Wu (2006) and Tsai and Tseng (2007).

```
01 Begin
02   Initialize population
03   Repeat until (termination condition(GA) = true) do
04     Select parents
05     Generate offspring population
06     For each individual in offspring population
07       Generate initial solution /* phase 2 begins */
08       Set initial temperature
09       Repeat until (termination condition(SA) = true) do
10         Generate a new solution
11         Let ΔE = fitness(new)-fitness(best-so-far)
12         If ΔE < 0, then best-so-far solution = new solution.
13         If ΔE ≥ 0, then accept new solution with probability AP.
14         Update temperature.
15       EndRepeat /* phase 2 ends */
16     EndFor
17   EndRepeat
18 End
```

Figure 3. Pseudo code of GA-SA algorithm.

In this algorithm, GA algorithm is applied to solve the job splitting problem (phase 1). A solution in this phase represents a set of sub-jobs. Its fitness value is the total tardiness of the considered parallel machine problem with this set of sub-jobs and is determined by using SA algorithm. After a new set of solutions (offspring population) is generated by GA through crossover and mutation in phase 1 (line 5 in the pseudo code), the procedure enters phase 2 where SA (line 7 to line 15) is applied to determine the fitness value of each individual in the offspring population. In phase 2, a solution represents the assignment of the sub-jobs (given by an individual from phase 1) to the machines and the job sequence on each machine. The outcome (the fitness value of each individual) is fed back to GA in phase 1 for generating the next offspring population. This procedure repeats until the termination condition of GA is met and the solution (a set of sub-jobs, how they are assigned to machines and job sequence on each machine) with the best total tardiness is obtained.

Figure 4 shows the pseudo code of SA-GA algorithm. In phase 1, SA is used to find the best set of sub-jobs. When a new solution is generated in this phase, GA in phase 2 is called for determining its fitness value.

The output of GA is then provided to SA for generation of the solution.

```
01 Begin
02   Generate initial solution
03   Set initial temperature
04   Repeat until (termination condition(SA) = true) do
05     Generate a new solution
06     Initialize population
07     Repeat until (termination condition(GA) = true) do
08       Calculate fitness values
09       Select parents
10       Generate offspring population
11     EndRepeat
12     Let ΔE = fitness(new)-fitness(best-so-far)
13     If ΔE < 0, then best-so-far solution = new solution.
14     If ΔE ≥ 0, then accept new solution with probability AP.
15     Update temperature.
16   EndRepeat
17 End
```

Figure 4. Pseudo code of SA-GA algorithm.

The other two-phase algorithms, SA-SA and GA-GA, are developed with the same concept.

4. TEST INSTANCES

Test instances are created for the purpose of evaluating the performance of the proposed heuristic algorithms. Instances may have different degrees of complexity due to how jobs are allowed to be divided. Generally speaking, instances with more jobs, more jobs consisting of larger number of stages or larger total number of job stages are more complex and require more efforts to find better solutions. To rank the complexity of test instances, a complexity index is developed. Let N denote the number of jobs and N_i denote the number of jobs with i stages. The maximum number of stages of a job in the considered group of jobs is denoted by $S_{\max}$. Furthermore, define

$$S = \frac{\sum_{i=1}^{S_{\max}} i \times N_i}{N}, \tag{1}$$

$$T = \frac{\sum_{i=1}^{S_{\max}} 2^{i-1} \times N_i}{N}, \tag{2}$$

$$T_{\max} = 2^{S_{\max}-1}. \tag{3}$$

The degree of instance complexity is defined as follows.

$$C = \frac{S \times T}{S_{\max} \times T_{\max}} \tag{4}$$

Clearly, $0 < C \leq 1$ and larger C values represent more complex instances. In this study, instances are categorized into three complexity levels. It is defined that the complexity level of an instance is CL-1 if $0 < C < 1/3$. If $1/3 \leq C < 1$, the complexity level is CL-2 and if $1/3 \leq C \leq 1$, the complexity level is CL-3.

For example, an instance contains 100 jobs. 20 jobs have only one stage. 30 jobs contain 2 stages. There are 20 jobs with 3 stages and the number of jobs with 4 stages is 30. Thus, $S_{\max} = 4$, $T_{\max} = 24 - 1 = 8$, and

$$S = \frac{1(20) + 2(30) + 3(20) + 4(30)}{100} = 2.6$$

$$T = \frac{1(20) + 2(30) + 4(20) + 8(30)}{100} = 4$$

Therefore,

$$C = \frac{2.6(4)}{4(8)} = 0.325$$

The complexity level of this instance is CL-1.

Three groups of test instances with various numbers of jobs and machines are generated. Table 1 shows the detail settings of the three instance groups. In each group, 10 instances in one of the three complexity levels are generated. Therefore, each group contains 30 instances and an overall of 90 instances are created. The maximum number of cut points in a job is set at 4. Processing time, cut points and due date of each job are randomly generated.

Table 1. Groups of test instance.

	Instance size		
	small	medium	large
Number of jobs	20	50	100
Number of machines	3	6	15

5. NUMERICAL EXPERIMENT

Four heuristic algorithms consisting of 2 phases are developed. Numerical experiment is conducted in order to evaluate their performance. For comparison, algorithms that do not consider job splitting and algorithms with jobs divided into the most possible number of sub-jobs are also developed. Again, SA and GA are applied in these single-phase algorithms. As a result, there are four more heuristic algorithms. The algorithm denoted as SA-no applies SA and does not consider any job splitting. Similarly, GA-no utilizes GA without any job splitting. The algorithms denoted as SA-all and GA-all use SA and GA, respectively, to schedule sub-jobs that can possibly be obtained.

A total of eight heuristic algorithms are developed using Devcpp (Dev C++) 4.9.9.2. The numerical ex-

periment is conducted on a computer system equipped with Intel Core2 Duo E8200 2.6GHz and 2GB RAM. The eight algorithms are applied to 90 test instances with various sizes and complexity levels. For each instance, 30 runs are executed with each algorithm. Total tardiness of each obtained solution is evaluated and run time is recorded.

5.1 Small-sized instances

For small-sized instances, algorithm SA-all obtains the best solution that can be found in the conducted experiment in every instance. Therefore, the percentage difference of the objective values obtained the other algorithms compared to the values obtained by SA-all are calculated and the results are listed in Table 2. Naturally, the average percentage difference of SA-all under all complexity levels is 0%. As can be seen, GA-all gives very low average percentage difference (1.21%). By dividing jobs into the most number of allowable subjobs, both SA-all and GA-all are able to explore more possible solutions and thus produces better solutions. SA-no and GA-no do not allow any job splitting and hence limit their search. As a result, their performance is the worst (8.26% and 10.53%). The four two-phase heuristic algorithms provide relatively low percentage difference, ranging from 2.56% to 4.33%. It demonstrates the benefits of taking advantage of the job splitting. The result also clearly shows that the percentages increase with higher degrees of complexity, independent of the algorithms.

Table 2. Average % difference(small-sized).

	SA-all	GA-all	SA-no	GA-no
CL-1	0.00%	0.80%	5.36%	8.68%
CL-2	0.00%	1.22%	8.32%	10.28%
CL-3	0.00%	1.60%	11.10%	12.61%
Average	0.00%	1.21%	8.26%	10.53%
	SA-SA	GA-SA	SA-GA	GA-GA
CL-1	2.49%	1.21%	3.55%	3.12%
CL-2	3.08%	2.81%	3.98%	3.71%
CL-3	4.95%	3.67%	5.46%	4.59%
Average	3.50%	2.56%	4.33%	3.81%

Table 3 shows the average CPU times (in seconds) of each heuristic algorithm. SA-all and GA-all require a lot greater CUP time for searching solutions (8.85 and 8.39 seconds on average, respectively) because they explore larger search space. SA-no and GA-no, on the other hand, take less than 1 second of CPU time (0.18 and 0.51 seconds on average, respectively). The four two-phase algorithms also show great search efficiency. SA-SA takes an average of only 1.35 seconds of CUP time and GA-GA needs less than 4 seconds on average.

Table 3. Average CPU time(small-sized).

	SA-all	GA-all	SA-no	GA-no
CL-1	6.1	5.63	0.18	0.51
CL-2	9.06	8.6	0.18	0.51
CL-3	11.4	10.94	0.18	0.51
Average	8.85	8.39	0.18	0.51
	SA-SA	GA-SA	SA-GA	GA-GA
CL-1	1.09	1.73	2.25	2.66
CL-2	1.453	2.44	2.98	3.75
CL-3	1.5	2.82	3.16	4.43
Average	1.35	2.33	2.80	3.61

Not surprisingly, it takes every heuristic algorithms longer time to solve instances with higher complexity levels. Noticeably, SA-all and GA-all are more sensitive to the complexity levels of instances as their average CPU times go up faster when complexity level increases.

Table 4 lists the average of the standard deviation (s.t.d.) of the objective values from the 30 runs for an instance using each algorithm on the three groups of instances with different complexity levels. The two algorithms, SA-no and GA-no, solve problems without the option of job splitting and their average standard deviations are considerably lower. It is due to the effective numbers of jobs to be scheduled are smaller in their cases. Over all, two-phase algorithms have higher average standard deviations. In addition, average standard deviation increases in the degree of complexity. With more complex instances, there are more jobs and machines to be considered. Consequently, the differences between the results of the 30 runs become greater.

Table 4. Average standard deviation(small-sized).

	SA-all	GA-all	SA-no	GA-no
CL-1	5.81	7.24	2.26	3.32
CL-2	5.96	7.63	2.47	3.49
CL-3	10.43	12.26	5.91	6.60
Average	7.40	9.04	3.55	4.47
	SA-SA	GA-SA	SA-GA	GA-GA
CL-1	8.08	5.49	9.70	7.78
CL-2	8.78	5.83	9.97	7.83
CL-3	18.69	15.17	18.97	17.64
Average	11.85	8.83	12.88	11.08

Next, the performance of the four two-phase heuristic algorithms is compared. The average percentage difference, average CPU time and average standard deviation of each algorithm are summarized in Table 5. In terms of solution quality and stability, algorithm GA-SA has the best performance among the four algorithms as it has the lowest average percentage difference (2.56%) and the lowest average standard deviation (8.83). It also

has a relatively low average CPU time (2.33 seconds). The algorithm with the lowest average CPU time is SA-SA. However, it also has the worst solution quality.

Table 5. Comparison of two-phase algorithms(small-sized).

	SA-SA	GA-SA	SA-GA	GA-GA
% difference	3.50%	2.56%	4.33%	3.81%
CPU time	1.35	2.33	2.80	3.61
s.t.d.	11.85	8.83	12.88	11.08

The superiority of algorithm GA-SA in solution quality can also be observed by comparing the numbers of instances where the algorithm obtains the best known solutions in the conducted experiment. The numbers are listed in Table 6. Among the 90 tested small-sized instances, GA-SA is able to find the best solution in 19 of them. It is far better than the other three algorithms, each of who finds the best known solution in no more than 5 instances.

Lastly, comparison is made on whether SA or GA is more suitable for phase 1 and phase 2 in a two-phase algorithm. Average percentage differences are listed in Table 7. SA-SA and GA-SA, both with SA in phase 2, are firstly compared. It can be seen that GA-SA performs better with an average percentage difference of 2.56%, compared to 3.50% of SA-SA. It indicates that GA is a better choice to be used for finding proper job splitting in phase 1 with respect to SA. Similarly, the comparison of SA-GA and GA-GA where both having GA in phase 2 shows that GA is more suitable for phase 1, as GA-GA has lower percentage difference (3.81% over 4.33%). Similarly, by comparing SA-SA and SA-GA, as well as GA-SA and GA-GA, one can find that SA is the better method for phase 2.

Table 6. Number of instances where best solutions are obtained(small-sized).

	SA-SA	GA-SA	SA-GA	GA-GA
CL-1	2	8	0	0
CL-2	2	6	0	2
CL-3	1	5	2	2
Total	5	19	2	4

Table 7. Comparison of SA and GA(small-sized).

Phase to be compared	Phase fixed	Method to be compared	
		SA	GA
Phase 1	Phase 2: SA	SA-SA	GA-SA
		3.50%	2.56%
	Phase 2: GA	SA-GA	GA-GA
		4.33%	3.81%
Phase 2	Phase 1: SA	SA-SA	SA-GA
		3.50%	4.33%
	Phase 1: GA	GA-SA	GA-GA
		2.56%	3.81%

5.2 Medium-sized instances

The obtained experimental results on the group of 30 medium-sized instances are analyzed. Average percentage differences of the obtained total tardiness with respect to the best total tardiness found in this experiment are summarized in Table 8. Again, SA-all has 0 average percentage differences in every complexity level. It indicates that the solutions provided by this algorithm is the best known solution in this experiment in each instance. All the other seven algorithms have greater percentage differences compared to those in the experiment with small-sized instances. Solution qualities of the two no-job-splitting algorithms, SA-no and GA-no, are again the worst. GA-all has the second lowest average percentage difference of 3.57%. It is followed by GA-SA with 4.44%.

However, it is reversed when comparing the algorithm performance in terms of search efficiency. From the average CUP time illustrated in Table 9, it can been observed that SA-all and GA-all have greater average CUP times (over 1 minute and 1.5 minutes, respectively). In contrast, SA-no takes only 0.38 second on average to solve a medium-sized instance and GA-no takes about 1 second. Comparatively, the four proposed two-phase heuristic algorithms are more balanced between solution quality and efficiency. Among them, SA-SA has the shortest average CUP time, followed by GA-SA. Over all, it takes longer to solve medium-sized instances.

Table 8. Average % difference(medium-sized).

	SA-all	GA-all	SA-no	GA-no
CL-1	0.00%	2.77%	11.25%	14.05%
CL-2	0.00%	3.51%	13.45%	15.14%
CL-3	0.00%	4.42%	13.72%	16.25%
Average	0.00%	3.57%	12.81%	15.15%
	SA-SA	GA-SA	SA-GA	GA-GA
CL-1	6.75%	2.78%	2.50%	3.49%
CL-2	9.15%	4.15%	7.22%	7.12%
CL-3	10.52%	6.38%	9.57%	8.26%
Average	8.80%	4.44%	6.43%	6.29%

Table 9. Average CPU time(medium-sized).

	SA-all	GA-all	SA-no	GA-no
CL-1	40.53	60.25	0.38	1.03
CL-2	60.72	90.38	0.38	1.03
CL-3	80.43	120.03	0.38	1.03
Average	60.56	90.22	0.38	1.03
	SA-SA	GA-SA	SA-GA	GA-GA
CL-1	8.75	13.25	18.47	21.62
CL-2	12.00	18.62	25.75	32.00
CL-3	12.38	21.00	26.29	37.20
Average	11.04	17.62	23.50	30.27

Table 10 compares the four proposed two-phase heuristic algorithms in average percentage difference, average CPU time and average standard deviation. SA-SA has the shortest average CPU time of only 11.04 seconds. However, its 8.80% average difference is the highest. Again, GA-SA has the best solution quality and stability, and the second shortest average CPU time among the four algorithms.

Solution quality of the algorithms can also be compared based on the number of instances where the algorithm obtains the best solutions found in the experiment. The numbers are shown in Table 11. The outcome shows that SA-SA fails to find best solution in any of the 90 medium-sized instances. In contrast, GA-SA obtains the best solution in 16 instances.

5.3 Large-sized instances

Experiment on large-sized instances is also conducted. Table 12 lists the average percentage differences of the obtained total tardiness with respect to the best total tardiness found in this experiment in the group of large-sized instances. Results similar to those obtained in the groups of small-sized and medium-sized instances are found. SA-all obtains the best known solution in every instance. Noticeably, the average percentage difference becomes larger in general with large-sized instances.

As expected, the CPU times required to solve large-sized instances become larger, as shown in Table 13. It takes an average of 460 seconds for SA-all to solve instances with the highest complexity level and almost 10 minutes for GA-all to do the same. In the mean time, the four two-phase algorithms also need larger CUP times but are a lot lower compared to SA-all and GA-all. It clearly demonstrates that when the numbers of jobs and machines are considerable large, two-phase algorithms are the more practical choices in balancing solution quality and efficiency. Surprisingly, SA-no still needs an average CPU time of no more than 1 second and GA-no requires no longer than 2 seconds on average.

As shown in Table 14 and Table 15, SA-SA does not perform well in terms of average percentage difference and number of instances where the best known solutions are found. Even though it has the smallest average CPU time of less than 1 minute, the solution quality is the worst among the four two-phase algorithms. Once again, GA-SA performs well in solution quality, stability and search efficiency.

Table 10. Comparison of two-phase algorithms (medium-sized).

	SA-SA	GA-SA	SA-GA	GA-GA
% difference	8.80%	4.44%	6.43%	6.29%
CPU time	11.04	17.62	23.50	30.27
s.t.d.	11.60	9.07	12.49	10.88

Table 11. Number of instances where best solutions are obtained (medium-sized).

	SA-SA	GA-SA	SA-GA	GA-GA
CL-1	0	3	5	2
CL-2	0	5	3	2
CL-3	0	8	0	2
Total	0	16	8	6

Table 12. Average % difference(large-sized).

	SA-all	GA-all	SA-no	GA-no
CL-1	0.00%	3.79%	14.88%	17.58%
CL-2	0.00%	4.53%	17.64%	20.46%
CL-3	0.00%	6.39%	18.31%	21.41%
Average	0.00%	4.90%	16.94%	19.81%
	SA-SA	GA-SA	SA-GA	GA-GA
CL-1	7.26%	4.68%	4.50%	4.26%
CL-2	10.87%	7.07%	8.36%	8.09%
CL-3	13.57%	8.35%	10.66%	9.45%
Average	10.57%	6.70%	7.84%	7.26%

Table 13. Average CPU time(large-sized).

	SA-all	GA-all	SA-no	GA-no
CL-1	218.5	273.0	0.65	1.78
CL-2	305.0	382.5	0.65	1.78
CL-3	460.5	594.0	0.65	1.78
Average	328.0	416.5	0.65	1.78
	SA-SA	GA-SA	SA-GA	GA-GA
CL-1	46.24	68.72	93.76	108.72
CL-2	61.28	95.64	130.60	156.24
CL-3	67.52	120.60	141.84	186.84
Average	58.35	94.99	122.07	150.60

Table 14. Comparison of two-phase algorithms(large-sized).

	SA-SA	GA-SA	SA-GA	GA-GA
% difference	10.57%	6.70%	7.84%	7.26%
CPU time	58.35	94.99	122.07	150.60
s.t.d.	11.65	9.03	12.82	10.98

Table 15. Number of instances where best solutions are obtained(large-sized).

	SA-SA	GA-SA	SA-GA	GA-GA
CL-1	0	2	5	3
CL-2	0	7	1	2
CL-3	0	7	0	3
Total	0	16	6	8

5.4 Comparison of two-phase algorithms

Next, the overall performance of the four two-phase algorithms is compared. Figure 5 illustrates average percentage difference of each heuristic algorithm in the three instance groups. It clearly shows that GA-SA has the best performance among the four algorithms. SA-SA provides the solutions with the worst quality for larger instances. In addition, average percentage differences go up as instances become larger no matter what algorithm is used. The number of instances where the best solution is obtained is summed up in Table 16. Note that there are 30 instances in each size. As shown in the table, GA-SA has overwhelming better performance over the other three algorithms in all instance sizes. It is able to obtain the best solution known in this study in over 56% of the tested instances. In contrast, the other three algorithms perform poorly under this category.

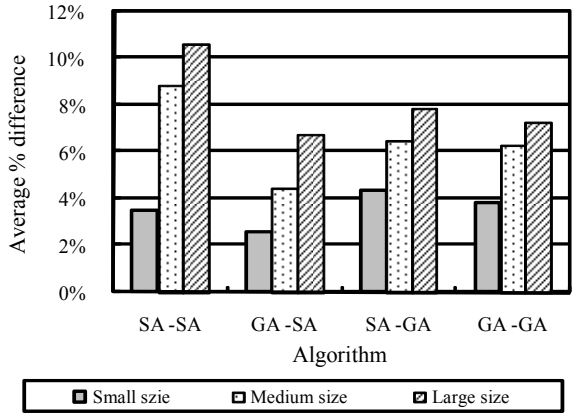

Figure 5. Algorithms vs. instance sizes in average % difference.

Table 16. Number of instances where best solutions are obtained.

Instance size	SA-SA	GA-SA	SA-GA	GA-GA
Small	5	19	2	4
Medium	0	16	8	6
Large	0	16	6	8
Total	5	51	16	18

Figure 6 provides the comparison in terms of average CPU time. SA-SA is most efficient in obtaining solutions and GA-SA has the second shortest average CPU time. As can be expected, it takes longer CPU time to solve larger instances regardless the algorithm used. Notice that CPU time increases drastically for obtaining solutions in large-sized instances as the number of jobs becomes 100 and the number of machines becomes 15. Comparison is also made in terms of the average of the standard deviation of the objective values from the 30 runs for an instance. It is shown in Figure 7. With the relatively smaller average standard deviation, GA-SA is superior in terms of robustness over the other three algorithms. Furthermore, the average standard deviation is very close in the three instance groups under either algorithm.

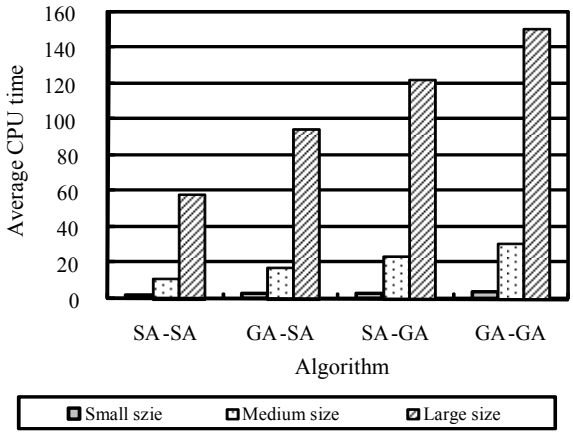

Figure 6. Algorithms vs. instance sizes in average CPU time.

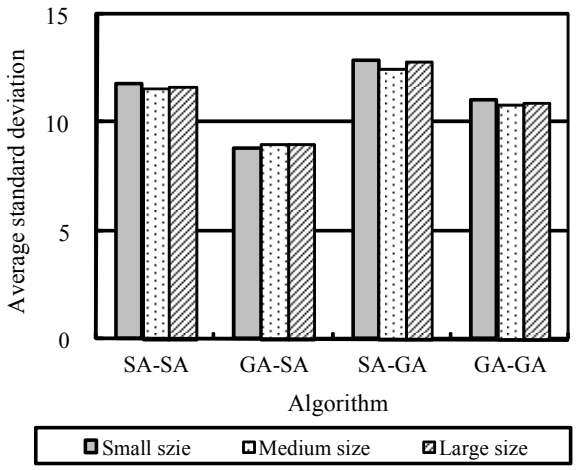

Figure 7. Algorithms vs. instance sizes in average standard deviation.

6. CONCLUSION

This paper studied parallel machine scheduling problems with options to divide jobs into multiple subjobs with pre-determined cut points. To utilize the characteristic of dividable jobs for finding better solutions and keep search time under control, a two-phase heuristic algorithm design is proposed. Simulated annealing and genetic algorithms are applied to the two algorithm phases. Accordingly, four heuristic algorithms were developed. Numerical experiment with test instances of various sizes and complexity levels were conducted. The analytical results led to the following conclusion.

1. Better solutions can be obtained by properly splitting jobs. Scheduling flexibility is increased if

jobs are divided into smaller sub-jobs. It produces more and potentially better solutions.

2. Dividing jobs into the greatest number of sub-jobs possible increases search time dramatically. It becomes considerably worse with greater number of jobs. The proposed two-phase algorithms can keep a balance between solution quality and search efficiency. They are suitable and more practical for larger-sized instances.
3. Both SA and GA can be applied for the two phases of the proposed algorithm. GA performs better in phase 1 as it is able to explore more effectively possible combination of job splitting. SA, on the other hand, is more efficient in phase 2 when determine job assignment and sequences, given a group of sub-jobs. As shown, algorithm GA-SA had the best overall performance by jointly considering solution quality, efficiency and robustness.

REFERENCES

Anagnostopoulos, G. C. and Rabadi, G. (2002). A simulated annealing algorithm for the unrelated parallel machine scheduling problem, *Proceedings of the 5th Biannual World Automation Congress*, **14**, 115-120.

Armentano, V. A. and de França Filho, M. F. (2007), Minimizing total tardiness in parallel machine scheduling with setup times: An adaptive memory–based GRASP approach, *European Journal of Operational Research*, **183**(1), 100-114.

Azizoglu, M. and Kirca, O. (1998), Tardiness minimization on parallel machines, *International Journal of Production Economics*, **55**(2), 163-168.

Azizoglu, M. and Kirca, O. (1999), On the minimization of total weighted flow time with identical and uniform parallel machines, *European Journal of Operational Research*, **113**(1), 91-100.

Bilge, U., Kıraç, F., Kurtulan, M., and Pekgün, P. (2004), A tabu search algorithm for parallel machine total tardiness problem, *Computers and Operations Research*, **31**(3), 397-414.

Cheng, R., Gen, M., and Tozawa, T. (1995), Minmax earliness/tardiness scheduling in identical parallel machine system using genetic algorithms, *Computers and Industrial Engineering*, **29**(1-4), 397-414.

Dessouky, M. M., Dessouky, M. I., and Verma, S. K. (1998), Flowshop scheduling with identical jobs and uniform parallel machines, *European Journal of Operational Research*, **109**(3), 620-631.

Guo, Y., Lim, A., Rodrigues, B., and Liang, Y. (2007), Minimizing the makespan for unrelated parallel machines, *International Journal on Artificial Intelligence Tools*, **16**(3), 309-415.

Kim, D., Kim, K., 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.

Kim, D., Na, D., 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.

Kim, D., Na, D., Jang, W., and Chen, F. F. (2006), Simulated annealing and genetic algorithm for unrelated parallel machine scheduling considering setup times, *International Journal of Computer Applications in Technology*, **26**(1-2), 28-36.

Liao, C., and Lin, C. (2003), Makespan minimization for two uniform parallel machines, *International Journal of Production Economics*, **84**(2), 205-213.

Low, C. (2005), Simulated annealing heuristic for flow shop scheduling problems with unrelated parallel machines, *Computers and Operations Research*, **32** (8), 2013-2025.

Mason, A. J. (1992), Genetic algorithm and scheduling problems, *Ph.D. thesis*, *Department of Management Sciences, University of Cambridge, UK*.

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**(4), 399-403.

Piersma, N. and Van Dijk, W. (1996), A local search heuristic for unrelated parallel machine scheduling with efficient neighborhood search, *Mathematical and Computer Modelling*, **24**(9), 11-19.

Rubin, P. A. and Ragatz, G. L. (1994), Scheduling in a sequence dependent setup environment with genetic search, *Computers and Operations Research*, **22**(1), 85-99.

Tan, K., Narasimhan, R., Rubin, P. A., and Ragatz, G. L. (2000), A comparison of four methods for minimizing total tardiness on a single processor with sequence dependent setup times, *Omega*, **28(3)**, 313-326.

Tsai, C.-Y. and Tseng, C.-J. (2007), Unrelated parallel-machines scheduling with constrained resources and sequence-dependent setup time, *Proceedings of the 37th International Conference on Computers and Industrial Engineering, Alexandria, Egypt*, 20-23.

Tsai, C.-Y. and Wu, S.-N. (2006), Application of simulated annealing algorithm on the unrelated parallel machine scheduling problem with limited resources, *Proceedings of the 36th International Conference on Computers and Industrial Engineering, Taipei, Taiwan*, 20-26.

Yalaoui, F., and Chu, C. (2002), Parallel machine scheduling to minimize total tardiness, *International Journal of Production Economics*, **76**(3), 265-279.