Multiprocessor task scheduling in multistage hybrid flow-shops: a genetic algorithm approach

F Sivrikaya Şerifoğlu[1]* and G Ulusoy[2]

[1]*Abant Izzet Baysal University, Bolu, Turkey;* [2]*Sabanci University, Istanbul, Turkey*

This paper considers multiprocessor task scheduling in a multistage hybrid flow-shop environment. The objective is to minimize the make-span, that is, the completion time of all the tasks in the last stage. This problem is of practical interest in the textile and process industries. A genetic algorithm (GA) is developed to solve the problem. The GA is tested against a lower bound from the literature as well as against heuristic rules on a test bed comprising 400 problems with up to 100 jobs, 10 stages, and with up to five processors on each stage. For small problems, solutions found by the GA are compared to optimal solutions, which are obtained by total enumeration. For larger problems, optimum solutions are estimated by a statistical prediction technique. Computational results show that the GA is both effective and efficient for the current problem. Test problems are provided in a web site at www.benchmark.ibu.edu.tr/mpt-hfsp.

Keywords: multiprocessor tasks; hybrid flow-shops; make-span minimization; genetic algorithms

Introduction

Hybrid flow-shop scheduling problems combine the properties of flow-shop scheduling problems and parallel machine (processor) scheduling problems. In flow-shops, jobs visit the stages of the shop in the same order of machines, and there is one machine at each stage. In hybrid flow-shops, at each stage, there are one or more identical machines (processors) to process the tasks. This availability of more than one processor at any stage provides additional flexibility for production planning and enhances the reduction of the production lead-time. The optimization criterion is usually the minimization of the make-span.

The two-stage hybrid flow-shop scheduling problem is shown to be NP-hard by Gupta.[1] Hoogeveen *et al*[2] showed that the pre-emptive case is also NP-hard. Despite the discouraging theoretical results, hybrid flow-shop scheduling models have captured the attention of many researchers. They are of practical interest in manufacturing environments like in the textile and process industries.

Research on hybrid flow-shop scheduling concentrates mainly on two-stage problems.[3,4] Few researchers consider three-stage problems.[5,6] Recently, the multistage hybrid flow-shop scheduling problem is being addressed more frequently.[7,8] Portmann *et al*[9] provide a hybrid algorithm crossing branch and bound with genetic algorithms (GAs) to solve the multistage problem. Exact,[10] local search,[11] and simulation[12] approaches are available. Linn and Zhang[13] and Riane and Artiba[14] provide surveys on multistage regular and hybrid flow-shop problems. Due date-based criteria such as the minimization of the maximum lateness[15] or tardiness[16] and the minimization of the number of tardy jobs[17] have also been applied.

In all the above-cited research work, jobs require only a single processor to be processed. This restriction can be relaxed to allow for multiprocessor tasks. There are several surveys on multiprocessor task scheduling,[18–20] and various research work on the general,[21] more specific,[22–24] single-objective,[25] and multiobjective[26] formulations of the problem. Application of meta-heuristics to multiprocessor task scheduling is also becoming popular.[27]

This paper considers multiprocessor task scheduling in hybrid flow-shops, which is a relatively new area of research. Oğuz and Ercan[28] provide constructive algorithms for the unit processing time case and Oğuz *et al*[29] for the arbitrary processing time case.

Application of meta-heuristics to the problem of multiprocessor task scheduling in a hybrid flow-shop environment is even more recent. Oğuz *et al*.[30] develop a tabu search-based heuristic for the multiprocessor task scheduling in a multistage hybrid flow-shop. Oğuz and Cheung[31] provide a GA and Sivrikaya Şerifoğlu and Tiryaki[32] present a simulated annealing algorithm for the multistage problem.

In this paper, a GA approach to the problem of scheduling of multiprocessor tasks in a multistage hybrid flow-shop is presented. The GA is tested against a lower bound from the literature on a large test bed and against actual and estimated optimum solutions. In the subsequent sections, the problem is defined, the GA approach is

*Correspondence: F Sivrikaya Şerifoğlu, Department of Management, Abant Izzet Baysal University, Bolu, Turkey.
E-mail: serifoglu_f@ibu.edu.tr

discussed, and computational results are given. Concluding remarks are provided in the last section.

Problem definition

The problem is the scheduling of n jobs each with m tasks in a hybrid flow-shop with m stages. At each stage $i=1, \ldots, m$, one task of job j is performed. Each stage i consists of m_i identical parallel processors. For the processing at any stage i, job j requires $size[i,j]$ processors simultaneously. That is, $size[i,j]$ processors assigned to job j at stage i start processing it simultaneously and continue doing so for a period of time equal to the processing time requirement of job j at stage i, namely $p[i,j]$. Each processor can process only one job at a time, and the processors do not break down. All jobs are ready at the beginning of the scheduling period. Pre-emption is not allowed. The objective is to minimize the make-span, that is, the completion time of all the tasks in the last stage.

The genetic algorithm

GAs are developed by Holland[33] as artificial adaptive systems and are increasingly used to attack optimization problems. The GA developed in this work is based on a permutation representation of the n jobs. This permutation denotes the sequence of jobs to be considered for scheduling in the first stage. For the sequencing at any other stage $i=2, \ldots, m$, the jobs are ordered according to non-decreasing completion times at the immediately preceding stage $i-1$.

For the scheduling at any stage $i=1, \ldots, m$, the next job j in the sequence associated with that stage is assigned to the first available $size[i,j]$ processors simultaneously. This type of scheduling is also employed by other researchers.[30,34]

The objective function to be minimized is taken to be the percentage deviation of the make-span from the lower bound, that is, $z=100*(C_{\max}-LB)/LB$, where z denotes the objective function, LB the lower bound, and $C_{\max}$ denotes the make-span.

A lower bound is developed by Oğuz *et al*[30] and is given by formula (1), where the set of jobs is denoted by J and the set of stages is denoted by M:

$$LB_1 = \max_{i\in M}\left\{\min_{j\in J}\left\{\sum_{k=1}^{i-1} p[k,j]\right\} + \frac{1}{m_i}\sum_{j\in J} p[i,j]size[i,j] + \min_{j\in J}\left\{\sum_{k=i+1}^{m} p[k,j]\right\}\right\} \quad (1)$$

The results of GA runs employing LB_1 indicated that the performance of LB_1 is rather weak and the deviations of the GA solutions from LB_1 are relatively large. Oğuz[35] has provided an improved version of LB_1, which combines ideas introduced in two earlier papers,[29,30] and which is given by expression 2 below. Henceforth, LB will denote this lower bound in expression 2. The rationale of these lower bound formulations is given in the appendix.

$$LB = \max_{i\in M}\left\{\min_{j\in J}\left\{\sum_{k=1}^{i-1} p[k,j]\right\} + \max\left\{\left(\sum_{j\in A_i} p[i,j] + \left\lceil\frac{\sum_{j\in B_i}(p[i,j]size[i,j]}{m_i}\right\rceil\right), \frac{1}{m_i}\sum_{j\in J} p[i,j]size[i,j]\right\} + \min_{j\in J}\left\{\sum_{k=i+1}^{m} p[k,j]\right\}\right\} \quad (2)$$

where

$$A_i = \left(j \middle| size[i,j] > \frac{m_i}{2}\right) \text{ and}$$
$$B_i = \left\{j \middle| size[i,j] = \frac{m_i}{2}\right\}.$$

The objective function needs to be transformed into a fitness function to be maximized by the GA. In this application, a popular and effective transformation is employed. The fitness function is defined to be the deviation of the objective function from the maximum (hence worst) objective function value in the current population. In mathematical terms, $f=Z_{\max}-z$, where f denotes the fitness function and $Z_{\max}$ the largest percentage deviation value in the current generation.

The selection operator employed is the roulette wheel selection operator. For the mutation operation, the job exchange mutation and the job replacement mutation operators are tried. The former operator exchanges places of the jobs at two randomly selected positions, and the latter moves a randomly chosen job to a randomly chosen position. Since the job exchange mutation operator performed better for various settings of the other GA parameters, it is preferred to the replacement operator.

Two different crossover operators are experimented with. One is the two-point crossover (version 1) suggested by Murata *et al*[36] for GA applications to flow-shop scheduling problems. Here, the child inherits jobs of one parent positioned outside two randomly selected points in the same positions as they are. The remaining jobs are ordered in the mid-part of the child in the order of their appearance on the other parent chromosome. The other crossover operator is a uniform order-based crossover (UOBX) suggested by Davis[37] for permutation representations. UOBX combines the relative orderings of jobs on the two parent chromosomes in the two children. Randomly selected jobs from one of the parents are fixed on one child. The rest of the jobs are first ordered so that they are in the same order as they

appear on the other parent, and then they are fed into the gaps on the child in the new order. The process is repeated for the second child starting with the other parent. UOBX performed better for various settings of other GA parameters; therefore, this operator is utilized in the numerical study.

The initial population is generated randomly but is seeded with three chromosomes: one denoting the shortest processing time (SPT) sequence according to stage 1 processing times, an other denoting the longest processing time (LPT) sequence according to stage 1 processing times, and the last one denoting the shortest total processing time (STPT) sequence. When generating job sequences according to the SPT and LPT rules, the tiebreak rule is that the job with a smaller total processing time is preferred. The tiebreak rule employed when sequencing jobs according to the STPT rule is that the job with a smaller processing time in the first stage is preferred.

A number of preliminary experiments are performed to fine tune the parameters of the GA. Population sizes of 50, 75, and 100; number of generations of 200 and 400; various values for crossover and mutation probabilities from the ranges 0.65–1.00 and 0.15–1.00, respectively, are tried. As a result, the parameters of the GA are set as follows: the population size is set at 50, the number of generations at 400, and both the probabilities for crossover and mutation at 0.75. Elitist strategy is employed by reproducing two of the best chromosomes at each generation intact into the next generation. The GA is replicated 5 times for each problem instance. A replication is stopped before 400 generations are generated, if the objective function value of an individual equals *LB*.

An example problem

A small example problem with five jobs and two stages is considered ($n=5$, $m=2$). Figure 1 illustrates the hybrid flow-shop under consideration. There are four processors in the first stage (P_{11}, P_{12}, P_{13} and P_{14}) and two processors in the second one (P_{21} and P_{22}). Problem data are given in Table 1. According to Table 1, job 1, for example, is to be processed simultaneously on two processors in the first stage for 86 time units and on one processor in the second stage for 90 time units. *LB* for this problem is found to be 337 using expression 2.

Figure 2 illustrates the Gantt chart for the schedule represented by an example solution [1 4 3 2 5]. This sequence gives the job sequence to be considered when scheduling in the first stage. The sequence of jobs in the second stage is obtained by sorting the jobs according to their completion times in the first stage. This second sequence turns out to be [4 1 3 2 5].

The make-span associated with the solution is 376. The percentage deviation from the *LB* is 11.57%. This is a considerable deviation for a small problem like the one considered. The GA has been run on this problem, and in all the replications this same solution has been found. After totally enumerating all the solutions for the example

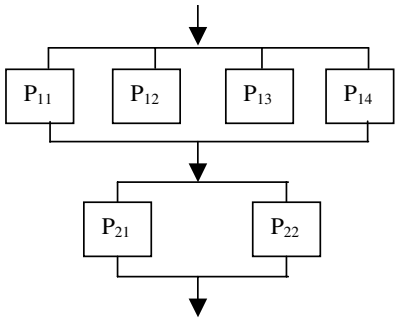

Figure 1 Illustration of the hybrid flow-shop in the two-stage example problem.

Table 1 Example problem data

Job j	*Stage i*	*Processing time p[i,j]*	*Processor requirement size[i,j]*
1	1	86	2
	2	90	1
2	1	99	4
	2	62	2
3	1	76	4
	2	94	1
4	1	14	1
	2	68	2
5	1	88	4
	2	27	2

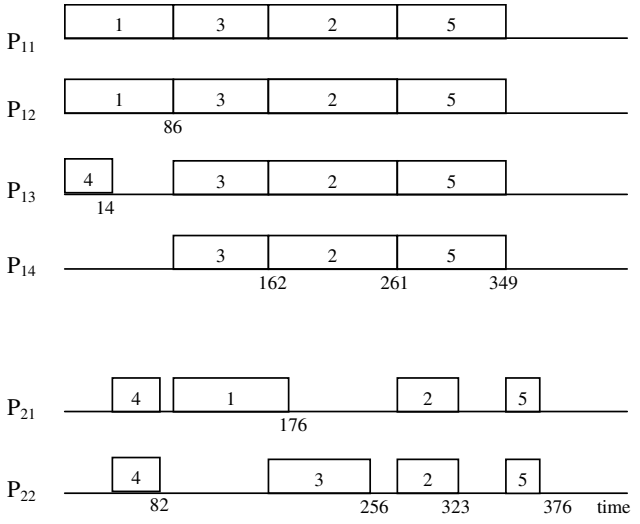

Figure 2 The Gantt chart for the schedule represented by the example chromosome [1 4 3 2 5] (not to scale).

problem, it has been verified that the solution found by the GA is, in fact, the optimal solution. More on this will be discussed in the next section on computational study.

Computational study

For comparison purpose, a similar experimental study as proposed by Oğuz *et al*[30] is employed. The number of jobs is taken to be $n = 5$, 10, 20, 50, 100 and the number of stages $m = 2$, 5, 8, 10. For each combination of n and m, 10 instances are generated for each of two types of problems, type-a and type-b. In type-a problems, the number of processors available at each stage m_i are allowed to vary and are determined randomly from the set $\{1, \ldots, 5\}$, whereas in type-b problems they are set equal to 5 for all stages, that is, $m_i = 5$ for $i = 1, \ldots, m$. In both cases, processor requirements of jobs, as given by $size[i,j]$, are determined randomly from the set $\{1, \ldots, m_i\}$. Processing time requirements $p[i,j]$ are taken to be integers and are determined randomly from the interval [1,100]. The test bed comprises 400 problems and is provided in the web site at www.benchmark.ibu.edu.tr/mpt-hfsp as well as in the *OR Library* at http://mscmga.ms.ic.ac.uk/info.html.

Comparison of the GA Solutions with the lower bound

The GA algorithm is replicated five times on each one of the 10 instances of type-a and type-b problems for each combination of n and m. The best solution from these five replications, that is, the solution with the make-span value that deviates least from the lower bound, is taken to be the GA solution for the corresponding problem. Table 2 gives the average percentage deviations of GA solutions from LB. The average is taken over the deviations corresponding to the 10 instances for each combination of n and m. Table 2 also gives the average run times per replication of GA in CPU seconds, including input and output operations where each replication of GA consists of up to 20 000 chromosome evaluations. The GA algorithm is compiled in Turbo Pascal 7.0 and is run on a computer with PIII processor of 1000 MHz and with 512 MB RAM. The run times for the GA are reasonable with at most 2.7 s per 1000 chromosome evaluations for 100-job problems.

From Table 2, it is observed that the deviations from *LB* are large, especially for small problems with five and 10 jobs. This may be rooted in the weak performance of the GA algorithm or in the inadequacy of *LB* or both. The lower bound has an important effect on the numerical results. In fact, the results reported in Table 2 are significantly better than the results obtained by employing the earlier version of the lower bound (LB_1) in the objective function evaluation within the GA. The results of these experiments employing LB_1 in the objective function formula in the GA are provided in Table 3. From a comparison of Tables 2 and 3, it can be deduced that the improvement in the lower bound resulted in improvements of the results of all the problems. The deviations for the problems with smaller numbers of jobs remain large even after the improvement in the lower bound, but the deviations for problems with larger numbers of jobs are significantly smaller.

Table 2 Average percentage deviations of solutions of GA (employing *LB*) from *LB* and average CPU times in seconds for type-a and type-b problems

		Type-a problems		*Type-b problems*	
n	m	*Avg%dev*	*CPU sec.*	*Avg%dev*	*CPU sec.*
5	2	6.67	0.38	10.74	0.44
	5	21.64	0.80	31.74	1.00
	8	24.27	1.00	24.44	1.18
	10	22.51	1.16	29.26	1.40
10	2	0.95	0.30	7.73	0.72
	5	9.12	1.10	14.98	1.56
	8	16.35	1.84	22.25	2.20
	10	13.35	2.08	17.95	2.62
20	2	0.92	0.76	2.38	1.42
	5	1.40	1.80	8.26	3.02
	8	5.71	3.58	12.08	4.54
	10	8.64	4.40	21.54	5.58
50	2	0.99	2.38	3.23	3.78
	5	1.57	5.60	10.30	9.18
	8	2.88	11.98	16.57	14.68
	10	2.95	15.52	17.92	18.38
100	2	0.45	5.72	1.86	9.08
	5	1.91	16.94	8.12	25.42
	8	1.73	33.14	10.90	48.50
	10	1.71	40.91	18.60	53.14

The results obtained are similar with respect to the magnitudes of deviations and their behaviour with increasing n and m to the ones obtained by Oğuz *et al*[30] employing their tabu search algorithm. The results are also similar in that much larger deviations are observed for type-b problems. In general, the deviations of the GA solutions from their respective lower bounds both in Tables 2 and 3 are smaller as compared to the deviations of the tabu search solutions; but since the test problems are different, it is hard to reach any conclusions as to the comparative performance of the two meta-heuristics.

According to the findings of Oğuz *et al*,[30] the average percentage deviations from the lower bound increase, in general, with increasing m and m_i values. They associate this with the degradation of the performance of the lower bound. The results reported in Tables 2 and 3 indicate that the average percentage deviation for a given n and m does indeed significantly increase, when m_i is increased to 5 in type-b problems. But for a given n, deviations do not always increase with increasing m. The effect of increasing m_i on

Table 3 Average percentage deviations of solutions of GA (employing LB_1) from LB_1 and average CPU times in seconds for type-a and type-b problems

		Type-a problems		*Type-b problems*	
n	*m*	*Avg%dev*	*CPU sec.*	*Avg%dev*	*CPU sec.*
5	2	9.26	0.50	24.89	0.62
	5	25.71	0.82	40.20	0.92
	8	26.30	1.00	32.42	1.20
	10	24.14	1.16	34.56	1.42
10	2	2.88	0.46	12.75	0.88
	5	10.45	1.12	19.58	1.58
	8	19.01	1.76	33.92	2.20
	10	13.68	2.08	30.75	2.64
20	2	5.46	1.20	6.17	1.50
	5	6.17	1.80	12.62	3.04
	8	5.89	3.60	25.20	4.58
	10	8.79	4.26	30.22	5.60
50	2	3.40	2.36	7.36	3.78
	5	2.56	5.56	15.84	9.24
	8	5.44	12.40	23.08	14.72
	10	4.43	15.40	26.58	18.38
100	2	3.31	5.82	9.34	9.08
	5	4.81	19.14	16.24	25.44
	8	2.95	31.88	20.89	41.86
	10	3.07	45.78	28.01	52.82

problem difficulty is quite significant as can be deduced by a comparison of deviations for type-a problems with that of type-b problems in the tables. The m_i values for type-a problems are generated from a uniform distribution. An investigation over the resulting distribution of m_i values employed in the experiments both for each n separately and over all n indicates that they are very close to the assumed distribution.

A limited study on two sets of additional test problems is conducted to see how the deviations change for problems with even larger m_i values. In the sets, there are 10 instances for each combination of $n = 5,10,20$ and $m = 2,5,8,10$ (and hence a total of 120 test problems in each one). In one set of problems $m_i = 6$ and in the other $m_i = 8$. The average percentage deviations from the lower bound for problems with $n = 20$ are consistently higher for problems with $m_i = 6$ and 8 as compared to deviations for problems with $m_i = 5$. For problems with $n = 10$, they are higher for problems with $m_i = 8$ but not for all problems with $m_i = 6$. And for problems with $n = 5$, there is again no consistent pattern. Overall the 120 problems, the average percentage deviation of average percentage deviations from the lower bound obtained for problems with $m_i = 6$ versus for problems with $m_i = 5$ is 24.98, With the same deviation for problems with $m_i = 8$ versus for problems with $m_i = 5$ is 39.17. In summary, it seems that m_i has an effect on problem difficulty especially for larger n; but this effect is also very much dependent on the data.

Oğuz *et al*[30] also note that the percentage deviations decrease with increasing n, which they find to be expected because of the increasing magnitudes of the make-span. Our findings indicate, in general, a similar trend, but we do associate it with the improving performance of the lower bound with increasing n. The assumptions underlying the lower bound formulae (no idle time and an even distribution of total work content on the processors) become more realistic with increasing n, since increasing the number of jobs increases the range of processing times and processor requirements and hence, increases possibilities for filling in the gaps on the processors.

The results obtained with the simulated annealing algorithm suggested by Sivrikaya Şerifoğlu and Tiryaki[32] indicate that their algorithm performs similarly as compared to the tabu search algorithm of Oğuz *et al*[30] and the GA algorithm proposed here, in that the performance degrades significantly with increasing problem difficulty. In fact, the degradation is worse for the simulated annealing algorithm for problems with number of jobs larger than 20. The researchers conclude that there is much room to improve the performance of the algorithm.

The experimental study conducted by Oğuz and Cheung[31] to test the performance of their GA algorithm is somewhat different. The researchers report overall average percentage deviations from the improved lower bound over problems with $n = 20$, 50, and 100. The average percentage deviation changes between 6.54 and 21.34 for problems with $m = 2$, between 9.21 and 16.35 for problems with $m = 5$, between 6.95 and 15.65 for problems with $m = 8$, and between 10.29 and 14.50 for problems with $m = 10$. The deviations seem to be higher than the ones given in Table 2, but again no conclusions can be drawn as the test problems are different and the deviations are not decomposed by Oğuz and Cheung[31] to reflect the effect of n.

To see whether the performance of the GA proposed in this study can be improved any further, the two-opt local search heuristic is integrated into the GA so that the best chromosome of each population is subjected to the two-opt heuristic. Solutions of the GA algorithm integrated with the two-opt heuristic are compared to the solutions of the GA algorithm without the two-opt heuristic. The results indicate that the integration of the two-opt heuristic does not lead to any statistically significant improvement. This may be due to the local improvements misleading the algorithm to false local peaks and hence, to a premature convergence. In addition, CPU times are prohibitively longer as compared with the CPU times resulting from the GA runs without the two-opt heuristic.

Alternatively, the two-opt heuristic is integrated into the GA so that only the best solution at the end of each replication is two-opted. This integration cannot lead to premature convergence; it refines solutions found by the GA

if possible. The results displayed in Table 4 indicate that two-opting does not provide any improvement at all for $n = 5$ and 10. The only statistically significant improvements are observed for $n = 50$ and 100, and especially for type-b problems. For smaller problems, the additional CPU time needed is minor but for larger problems it becomes significant as would be expected.

Comparison of the GA solutions with the optimal solutions for small problems

As noted above, the deviations of GA solutions from *LB* are large, especially for small problems with five and 10 jobs. Incorporation of the two-opt heuristic has not led to a decrease in the deviations. This leads to the hypothesis that the solutions found by the GA are already good solutions and it is the *LB* that remains weak.

To test this hypothesis, the GA solutions are compared to the optimal solutions for 5-job problems, which are obtained by the total enumeration of the job sequences for each one of the 80 problem instances. The results are presented in Table 5. GA is able to find the optimal solution for all problem instances. This means that the deviations reported in Tables 2 and 3 for $n = 5$ correspond to deviations of the optimal solutions from *LB* and from LB_1, respectively.

This result explains why for the case of 5-job problems, two-opting the best solution at the end of each replication has not led to any improvement. Although we do not have numerical evidence, it might be conjectured that the same reasoning holds for the case of 10-job problems, that is, GA solutions are either optimal or close to optimal.

Table 4 Comparison of solutions of GA with the two-opt heuristic to solutions of GA without the two-opt heuristic ($t_{9,0.01} = 2.821$; $t_{9,0.005} = 3.250$)

		Type-a problems		*Type-b problems*	
n	*m*	*Avg%dev*	*t-Value*	*Avg%dev*	*t-Value*
5	2	0.00	—	0.00	—
	5	0.00	—	0.00	—
	8	0.00	—	0.00	—
	10	0.00	—	0.00	—
10	2	0.00	—	0.00	—
	5	0.00	—	0.00	—
	8	0.00	—	0.00	—
	10	0.00	—	0.00	—
20	2	0.00	—	−0.07	1.86
	5	0.00	—	−0.18	1.25
	8	−0.11	1.00	−0.06	1.00
	10	−0.14	1.31	−0.22	1.41
50	2	−0.70	2.63	−1.17	4.12
	5	−0.65	1.75	−2.16	9.01
	8	−1.33	3.25	−2.42	5.59
	10	−0.81	2.79	−1.79	5.01
100	2	−0.99	3.34	−2.59	5.12
	5	−2.09	2.49	−4.41	17.09
	8	−1.27	3.31	−4.09	11.09
	10	−1.77	3.38	−3.80	9.20

Table 5 Average percentage deviations of GA solutions from the optimal solutions for 5-job problems

		Type-a problems		*Type-b problems*	
n	*m*	*Avg%dev*	*t-Value*	*Avg%dev*	*t-Value*
5	2	0.00	—	0.00	—
	5	0.00	—	0.00	—
	8	0.00	—	0.00	—
	10	0.00	—	0.00	—

Table 6 Average percentage deviations of GA solutions from the best of SPT, LPT, and STPT solutions and the corresponding *t*-values ($t_{9,0.01} = 2.821$; $t_{9,0.005} = 3.250$)

		Type-a problems		*Type-b problems*	
n	*m*	*Avg%dev*	*t-Value*	*Avg%dev*	*t-Value*
5	2	−5.22	2.89	−8.80	4.15
	5	−6.88	3.84	−6.93	4.56
	8	−7.77	3.60	−7.22	3.63
	10	−5.28	5.61	−7.19	3.88
10	2	−7.32	3.73	−16.91	7.40
	5	−14.98	7.77	−17.45	8.80
	8	−17.00	9.23	−13.54	9.09
	10	−14.59	12.27	−14.98	9.21
20	2	−12.72	4.92	−18.81	7.98
	5	−16.36	19.12	−21.74	11.49
	8	−15.02	10.06	−20.18	17.12
	10	−17.65	13.67	−18.06	17.71
50	2	−12.66	6.20	−17.35	25.99
	5	−11.57	6.30	−21.90	14.59
	8	−14.39	7.86	−19.69	15.82
	10	−15.22	10.93	−20.72	14.40
100	2	−8.82	5.00	−15.55	16.44
	5	−10.55	4.36	−20.13	24.51
	8	−12.06	6.80	−18.84	13.31
	10	−11.49	9.05	−19.40	22.03

Comparison of the GA solutions with the solutions of heuristic rules

GA solutions are compared to the best of the solutions provided by the heuristic rules (SPT, LPT, and STPT), which are employed to seed the initial population. The improvement brought about by the GA algorithm over these heuristic rules is thus illustrated. Table 6 gives the average percentage deviations of GA solutions from the best of solutions of these rules.

As the t-values reported in Table 6 indicate, the GA yields statistically significant improvements as compared to the heuristic rules. The relative performance of the GA improves with increasing m_i, and it first improves and then deteriorates with increasing n.

Comparison of GA solutions with the estimated optima

Encouraged by the successful comparisons of the GA solutions to the optimal values for the 5-job problems, optimum solutions for the other larger problems are sought. Rardin and Uzsoy[38] discuss the statistical estimation of optimal values for combinatorial optimization problems as a way to evaluate the performance of heuristics.

The basis of the estimation method applied here is a result by Fisher and Tippett[39] about the distribution of least values, which briefly states that the least of M random variables with a common distribution on real numbers greater than or equal to a is asymptotically distributed Weibull, with a being the location parameter. Objective function values for feasible solutions to an optimization problem can be thought of as points from an objective value distribution, and due to the very large number of such solution values for a combinatorial optimization problem, the continuity assumption can be approximately justified.

For the estimation process, K-independent samples each consisting of T_k objective values are obtained. Let z_i be the minimum value of sample i, $i=1,\ldots,K$. These sample minima, assumed to be independent, are sorted so that $z_{(1)}\leqslant z_{(2)}\leqslant\cdots\leqslant z_{(K)}$. As shown by Zanakis,[40] the location parameter a of the Weibull distribution can then be estimated as

$$\hat{a}=[z_{(1)}z_{(K)}-z_{(2)}^2]/[z_{(1)}+z_{(K)}-2z_{(2)}] \qquad (3)$$

This estimate of a also provides an estimate of the optimum solution value.

Ovacık *et al*[41] integrate such an estimation procedure with a simulated annealing heuristic and apply it on a single machine maximum lateness problem with sequence-dependent set-up times. Ghashghai and Rardin[42] make use of the solutions generated by their GA algorithm to estimate the optima for the problem of finding sub-graphs that meet survivability requirements.

Here, 20 sample optima z_i are obtained by two-opted SPT, LPT, STPT solutions, a set of 12 randomly generated and two-opted solutions and best solutions of five runs of a smaller GA (population size = 25, number of generations = 40). Repeated values among these z_i values are eliminated, and the rest of the z_i array is subjected to runs test (at a significance level of 0.05) to guarantee that the assumption of independent sample minima holds. Using expression (3), estimates for the optimum values are then obtained. The GA solutions are compared to these estimates, and the results are presented in Table 7 and Figure 3.

For a few problem instances, many or all the z_i values are the same and the runs test fails. For various other problems, even though there are many distinctly different z_i values, the runs test still fails. For 5-job problems, for instance, all runs tests fail; therefore, these problems are not included in Table 7. The column 'ncases' in Table 7 gives the number of problem instances among 10 for which the runs test is successful and an estimate can be obtained. In total, estimates are obtained for 280 from among 400 problems.

The results in Table 7 show that average percentage deviations of GA solutions from estimated optima are not statistically significant, except for three groups of problems with $n=20$ $m=8$, $n=20$ $m=10$, and $n=100$ $m=10$. For some problems, the GA solutions are slightly better than the

Table 7 Average percentage deviations of GA solutions from the estimated optima ($t_{1,0.01}=31.821$; $t_{4,0.01}=3.747$; $t_{5,0.01}=3.365$; $t_{6,0.01}=3.143$; $t_{7,0.01}=2.998$; $t_{8,0.01}=2.896$)

		Type-a problems			*Type-b problems*		
n	*m*	*Avg%dev*	*t-Value*	*ncases*	*Avg%dev*	*t-Value*	*ncases*
10	2	0.00	2.00	2	−0.31	0.59	7
	5	−1.08	2.28	9	−0.48	1.41	10
	8	−0.46	1.75	10	−0.67	1.87	10
	10	1.24	1.23	10	−0.39	0.98	10
20	2	0.01	0.24	8	−0.61	1.90	10
	5	−0.61	1.58	9	−1.49	2.74	10
	8	−1.65*	3.39	9	−1.49	1.91	10
	10	−1.41*	4.60	10	−1.18	2.30	10
50	2	0.14	0.57	5	0.13	1.30	10
	5	0.26	1.88	9	−0.67	1.91	10
	8	−0.21	0.83	8	−0.25	0.29	9
	10	−0.11	0.27	9	0.33	0.49	9
100	2	0.24	0.92	5	0.20	1.46	9
	5	−0.05	0.16	8	0.22	0.52	10
	8	0.04	0.00	6	−0.07	0.22	10
	10	0.45*	3.34	9	−0.54	1.47	10

*Statistically significant at 0.01 significance level.

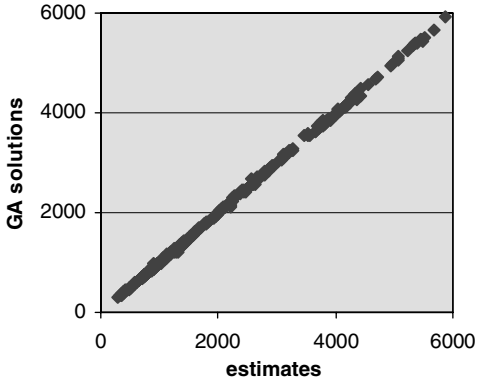

Figure 3 Scatter diagram of GA solutions versus estimated optima for 280 problems.

estimates, and the deviations are negative. The maximum deviation occurs for 20-job problems, and it is less than 1.70%.

Figure 3 illustrates a scatter diagram of the GA solutions versus estimated optima. As the deviations are small and mostly insignificant, the plotted values are concentrated along the diagonal.

Concluding remarks

The problem of multiprocessor task scheduling in a multi-stage hybrid flow-shop setting is addressed by means of a GA approach. GA results are compared with a lower bound and its improved version from the literature, with solutions obtained by SPT, LPT, and STPT heuristic rules, with optimum solutions for small problems, and with estimated optimum values for larger problems. The performance of the lower bound is found to be inadequate for problems involving small number of jobs and for problems involving large number of processors per stage. But its performance is observed to improve with increasing the number of jobs.

For possible further improvement, the two-opt local search heuristic is integrated into the GA so that the best chromosome of each population is subjected to the two-opt heuristic. The results indicate that the integration of the two-opt heuristic does not lead to any statistically significant improvement.

The GA is able to provide significant improvements over heuristic dispatching rules; it is able to find optimum solutions for small problems and near-optimum solutions for larger problems.

There is a need for better lower bounds and for more benchmark results. The test problems employed here are provided in a web site at www.benchmark.ibu.edu.tr/mpt-hfsp to facilitate interaction among current and potential researchers working on the same and similar subjects.

References

1 Gupta JND (1988). Two-stage hybrid flow-shop scheduling problem. *J Opl Res Soc* **39**: 359–364.
2 Hoogeveen JA, Lenstra JK and Veltman B (1996). Pre-emptive scheduling in a two-stage multiprocessor flow-shop is NP-hard. *Eur J Opl Res* **89**: 172–175.
3 Sundararaghavan PS, Kunnathur AS and Viswanathan I (1997). Minimizing make-span in parallel flow-shops. *J Opl Res Soc* **48**: 834–842.
4 Haouari M and M'Hallah R (1997). Heuristic algorithms for the two-stage hybrid flow-shop problem. *Opns Res Lett* **21**: 43–53.
5 Riane F, Artiba A and Elmaghraby SE (1998). A hybrid three-stage flow-shop problem: Efficient heuristics to minimize make-span. *Eur J Opl Res* **109**: 321–329.
6 Dessouky MM, Dessouky MI and Verma SK (1998). Flow-shop scheduling with identical jobs and uniform parallel machines. *Eur J Opl Res* **109**: 620–631.
7 Gupta JND *et al* (2002). Heuristics for hybrid flow shops with controllable processing times and assignable due dates. *Comput Opns Res* **29**: 1417–1439.
8 Riane F, Artiba A and Iassinovski S (2001). An integrated production planning and scheduling system for hybrid flowshop organizations. *Int J Production Econom* **74**: 33–48.
9 Portmann M-C, Vignier A, Dardilhac D and Dezalay D (1998). Branch and bound crossed with GA to solve hybrid flow-shops. *Eur J Opal Res* **107**: 389–400.
10 Moursli O and Pochet Y (2000). A branch-and-bound algorithm for the hybrid flow-shop. *Int J Production Econom* **64**: 113–125.
11 Negenman EG (2001). Local search algorithms for the multi-processor flow-shop scheduling problem. *Eur J Opl Res* **128**: 147–158.
12 Grangeon N, Tanguy A and Tchernev N (1999). Generic simulation model for hybrid flow-shop. *Comput Ind Eng* **37**: 207–210.
13 Linn R and Zhang W (1999). Hybrid flow shop scheduling: a survey. *Comput Ind Eng* **37**: 57–61.
14 Riane F and Artiba A (1999). Scheduling multistage flow-shop problem: a brief review. In: *Proceedings of the International Conference on Industrial Engineering and Production Management, Glasgow*, ISBN 2-930294-02-7, Volume 2. Facultés Universitaires Catholiques de Mons, Mons, Belgium, pp 323–335.
15 Botta-Genoulaz V (2000). Hybrid flow shop scheduling with precedence constraints and time lags to minimize maximum lateness. *Int J Production Econom* **64**: 101–111.
16 Vignier A, Billaut J-C and Proust C (1996). Minimizing maximum tardiness in some two-stage hybrid flow-shops. In: *Proceedings of the 5th International Workshop on Project Management and Scheduling*. Scientific Publishers OWN PAN, Poznan, Poland, pp 253–257.
17 Gupta JND and Tunc EA (1998). Minimizing tardy jobs in a two-stage hybrid flow-shop. *Int J Production Res* **36**: 2397–2417.
18 Brucker P and Kraemer A (1996). Polynomial algorithms for resource-constrained and multi-processor task scheduling problems. *Eur J Opl Res* **90**: 214–226.
19 Drozdowski M (1996). Scheduling multiprocessor tasks — an overview. *Eur J Opl Res* **94**: 215–230.
20 Lee C-Y, Lei L and Pinedo M (1997). Current trends in deterministic scheduling. *Ann Opns Res* **70**: 1–41.
21 Chen J and Chung-Yee L (1999). General multiprocessor task scheduling. *Naval Res Logistics* **46**: 57–74.
22 Amoura AK, Bampis E, Manoussakis Y and Tuza Z (1999). A comparison of heuristics for scheduling multiprocessor tasks on three dedicated processors. *Parallel Comput* **25**: 49–61.
23 Cai X, Lee C-Y and Li C-L (1998). Minimizing total flow time in multiprocessor task systems with pre-specified processor allocations. *Naval Res Logistics* **45**: 231–242.
24 Bianco L, Blazewicz J, Dell'Olmo P and Drozdowski M (1997). Linear algorithms for pre-emptive scheduling of multiprocessor tasks subject to minimal lateness. *Discrete Appl Math* **72**: 25–46.
25 Drozdowski M and Dell'Olmo P (2000). Scheduling multi-processor tasks for mean flow time criterion. *Comput Opns Res* **27**: 571–585.
26 Cai X, Wong T-L and Lee C-Y (2000). Multiprocessor task scheduling to minimize the maximum tardiness and the total completion time. *IEEE Trans Robotics Automat* **16**: 824–830.
27 Correa RC, Ferreira A and Rebreyend P (1999). Scheduling multiprocessor tasks with genetic algorithms. *IEEE Trans Parallel Distributed Syst* **10**: 825–837.
28 Oğuz C and Ercan MF (1997). Scheduling multiprocessor tasks in a two-stage flow-shop environment. *Comput Ind Eng* **33**: W269–W272.
29 Oğuz C, Ercan MF, Cheng TCE and Fung YF (2003). Heuristic algorithms for multiprocessor task scheduling in a two-stage hybrid flow-shop. *Eur J Opl Res* **149**: 390–403.

30 Oğuz C *et al*. (2004). Hybrid flow-shop scheduling problems with multiprocessor task systems. *Eur J Opl Res* **152**: 115–131.
31 Oğuz C and Cheung B (2002). A genetic algorithm for flow-shop scheduling problems with multiprocessor tasks. In: Valls V *et al* (eds). *Proceedings of the Eighth International Workshop on Project Management and Scheduling*. Fundacion Universidad-Empresa de Valencia, Valencia, Spain, pp 282–286.
32 Sivrikaya Şerifoğlu F and Tiryaki IU (2002). Multiprocessor task scheduling in multistage hybrid flow-shops: A simulated annealing approach. In: Baykasoglu A and Develi T (eds). *Proceedings of the 2nd International Conference on Responsive Manufacturing*. University of Gaziantep Printing Office, Gaziantep, Turkey, pp 270–274.
33 Holland J (1975). *Adaptation in Natural and Artificial Systems*. The University of Michigan Press: Ann Arbor.
34 Riane F, Raczy C and Artiba A (1999). Hybrid auto-adaptable simulated annealing based heuristic. *Comput Ind Eng* **37**: 277–280.
35 Oğuz C (September 2002). *Personal communication*.
36 Murata T, Ishibuchi H and Tanaka H (1996). Genetic algorithms for flow-shop scheduling problems. *Comput Ind Eng* **30**: 1061–1071.
37 Davis L (1991). *Handbook of Genetic Algorithms*. Van Nostrand Reinhold: New York.
38 Rardin RL and Uzsoy R (2001). Experimental evaluation of heuristic optimisation algorithms: a tutorial. *J Heuristics* **7**: 261–304.
39 Fisher R and Tippett L (1928). Limiting forms of the frequency distribution of the largest or smallest member of a sample. *Proc Cambridge Philos Soc* **24**: 180–190.
40 Zanakis SH (1979). A simulation study of some simple estimators of the three-parameter Weibull distribution. *J Stat Comput Simulation* **9**: 419–428.
41 Ovacık IM, Rajagopalan S and Uzsoy R (2000). Integrating interval estimates of global optima and local search methods for combinatorial optimisation problems. *J Heuristics* **6**: 481–500.
42 Ghashghai E and Rardin RL (1998). *Using a hybrid of exact and genetic algorithms to design survivable networks*, Working paper School of Industrial Engineering, Purdue University.
43 Santos DL, Hunsucker JL and Deal DE (1995). Global lower bounds for flowshops with multiple processors. *Eur J Opl Res* **80**: 112–120.
44 Syslo MM, Deo N and Kowalik JS (1983). *Discrete Optimization Algorithms*. Prentice-Hall: Englewood Cliffs, NJ.

Appendix: The rationale of the lower bound formulations

The lower bound given in formula (1) is a stage-based lower bound similar to the lower bound suggested by Santos *et al*[43] for scheduling 'single-processor' jobs in hybrid flowshops. Here, associated with each stage i, $i=1,\ldots,m$, is a lower bound on the make-span, say $LB(i)$. The overall lower bound LB_1 is the maximum of these bounds, that is, $LB_1 = \max_{i\in M} LB(i)$, where $LB(i)$ is defined as follows:

$$LB(i) = \min_{j\in J}\left\{\sum_{k=1}^{i-1} p[k,j]\right\} + \frac{1}{m_i}\sum_{j\in J} p[i,j]size[i,j] + \min_{j\in J}\left\{\sum_{k=i+1}^{m} p[k,j]\right\}$$

The logic behind the proof for the $LB(i)$ formulation is similar to the one used in the proof provided by Santos *et al*[43] It is based on the assumption that no idle times occur on the processors throughout the duration of the schedule. Under this assumption, the time needed to start processing on any machine at stage i is at best equal to $\min_{j\in J}\{\sum_{k=1}^{i-1} p[k,j]\}$. The minimum time required to finish processing of the jobs at the remaining stages $i+1$ through to m is at best $\min_{j\in J}\{\sum_{k=i+1}^{m} p[k,j]\}$. The middle part of the $LB(i)$ formulation pertains to the bound on the duration of the processing of jobs at stage i. The minimum for the duration of the processing of jobs at stage i occurs when the constraints on the simultaneous processing on $size[i,j]$ processors are not respected and the jobs can be preempted as often as required to allow an even distribution of the total work content associated with stage i. The total work content for stage i is given by $\sum_{j\in J} p[i,j]size[i,j]$, and when evenly distributed over all the processors, it gives rise to a duration of $(1/m_i)\sum_{j\in J} p[i,j]size[i,j]$.

Looking at stage i from a different perspective, it can also be thought that there are $size[i,j]$ replicates of each job j, each with the duration $p[i,j]$, so that there are altogether $\sum_{j\in J} size[i,j]$ 'single-processor' jobs in a set J' to be scheduled on m_i processors at stage i. The make-span associated with stage i is then bounded from below by $\sum_{j\in J'} p[i,j]/m_i$. (see, eg, Syslo *et al*,[44] p 502). But $\sum_{j\in J'} p[i,j] = \sum_{j\in J} p[i,j]size[i,j]$, and this completes the proof of LB_1.

LB given in formula (2) differs from LB_1 by the refinement associated with the distribution of work content at stage i, $i=1,\ldots,m$. We can be sure that jobs with $size[i,j] > m_i/2$ (ie, jobs $j\in A_i$) will have at least one common processor, on which they all will be scheduled, and the best possible way to do this is that they are scheduled one after the other with no inserted idle time. This scheduling will give rise to a duration of magnitude $\sum_{j\in A_i} p[i,j]$ on that bottleneck processor.

From the rest of the jobs, jobs with $size[i,j] = m_i/2$ (ie, jobs $j\in B_i$) can best be scheduled such that their work content is distributed evenly on the processors. This gives rise to a duration of magnitude $(1/m_i)\sum_{j\in B_i} p[i,j]size[i,j] = \frac{1}{2}\sum_{j\in B_i} p[i,j]$.

Finally, jobs with $size[i,j] < m_i/2$ will, in the best case, be scheduled to fit in to the idle times on the processors not used by the set of jobs in A_i so that no idle time and hence no extension on the overall duration occurs.