International Journal of Mechanical, Industrial and Aerospace Sciences
ISSN: 2517-9950
Vol:5, No:7, 2011

A Hybrid Genetic Algorithm for the Sequence Dependent Flow-Shop Scheduling Problem

Mohammad Mirabi

***Abstract*—**Flow-shop scheduling problem (FSP) deals with the scheduling of a set of jobs that visit a set of machines in the same order. The FSP is NP-hard, which means that an efficient algorithm for solving the problem to optimality is unavailable. To meet the requirements on time and to minimize the make-span performance of large permutation flow-shop scheduling problems in which there are sequence dependent setup times on each machine, this paper develops one hybrid genetic algorithms (HGA). Proposed HGA apply a modified approach to generate population of initial chromosomes and also use an improved heuristic called the iterated swap procedure to improve initial solutions. Also the author uses three genetic operators to make good new offspring. The results are compared to some recently developed heuristics and computational experimental results show that the proposed HGA performs very competitively with respect to accuracy and efficiency of solution.

***Keywords*—**Hybrid genetic algorithm, Scheduling, Permutation flow-shop, Sequence dependent

I. Introduction

The flow-shop scheduling problems (FSP) have been studied for over five decades. The classical flow-shop problem with the make-span minimization criterion has always attracted the attention of researchers because of its applications in practice. For example, a typical ship builder builds a number of different ship models, but the individual parts that go into each ship follow similar processes. A shipyard can be thought of as a collection of several flow-shop s. In particular, almost all parts, i.e. the big metal blocks, go through a panel shop in a ship yard where they are cut or welded together. Panel shops are typically treated as flow-shop s. The flow-shop problem is easy to describe and formulate, yet computationally it is rather challenging. Therefore, this problem has inspired the development of a number of solution procedures [1].In an *m* machine flow shop, there are *m* stages in series, where there exist one or more machines at each stage. Each job has to be processed in each of the *m* stages in the same order. That is, each job has to be processed first in stage 1, then in stage 2, and so on. Processing times for each job in different stages may be different. We classify flow shop problems as (i) flow shop (there is one machine at each stage), (ii) no-wait flow shop (a succeeding operation starts immediately after the preceding operation completes), (iii) flexible (hybrid) flow shop (more than one machine exist in at least one stage), and (iv) assembly flow shop (each job consists of *m*-1 specific operations, each of which has to be performed on a pre-determined machine of the first stage, and an assembly operation to be performed on the second stage machine) [2].In the flow-shop literature, one can find an overwhelming number of papers for the regular flow-shop problem with the objective of minimizing the maximum completion time across all jobs (also called make-span and denoted by C_{max}). However, the Sequence Dependent Setup Time Flow-shop Problem (SDST flow-shop problem in short) has attracted much less attention especially before 2000. Specific to the SDST flow-shop are the setup times.The objective in flow-shop scheduling problems is to find a sequence for processing the jobs on the machines so that a given criterion is optimized. This yields a total of $n!$ possible orderings of the operations on each machine and a total of $(n!)m$ possible processing sequences. In flow-shop scheduling research usually only so called permutation sequences are considered, where the processing order of operations is the same for all machines. Here, the author also adopts this restriction. In this paper, the author considers sequence dependent flow-shop scheduling problem with the make-span minimization criterion.Regarding the computational complexity, the SDST flow-shop with the C_{max} objective has been shown to be NP-hard by Gupta [3] even when $m = 1$ and also when $m = 2$ and setups are present only on the first or second machine [4]. For $m = 1$, the SDST flow-shop is known to be a special case of the Traveling Salesman Problem (TSP) that is also well known to be NP-hard which means that an efficient algorithm for solving the problem to optimality is unavailable. Therefore, solving the problem by an exact algorithm is time consuming and computationally intractable. To deal with the problem efficiently and effectively, one hybrid genetic algorithms (HGA) is developed in this paper.The paper is organized as follows. Section 2 surveys on the relevant literature. Formulation is given in section 3. Section 4 discusses the principles of the algorithms used to solve the FSP. Section 5 compares the performance of the algorithms. Finally, Section 6 concludes the paper.

Mohammad Mirabi is with the Industrial Engineering Department, Islamic Azad University-Ashkezar Branch, Ashkezar, Iran (phone: (98)3523625955; fax: (98)3523625952; email: m.mirabi@yahoo.com).

II. Literature review

Compared to the regular flow-shop, on which hundreds of papers have been published, the literature on the SDST counterpart is scarce.

In the pioneering work of Johnson [5], the author proposed a simple rule to obtain optimal sequences for the permutation flow-shop problem (PFSP) with two machines. This work raised significant interest in the PFSP and was followed by several attempts for solving the PFSP with more than two machines. Due to the NP-completeness of the PFSP [6], researchers have mainly focused on the development of effective heuristics and meta-heuristics. Salient heuristics are due to Campbell et al. [7] (also called CDS method) and the well-known NEH heuristic by Nawaz et al. [8]. Some noteworthy meta-heuristics have also been proposed, for example the simulated annealing by Osman and Potts [9], the tabu search of Widmer and Hertz [10] and the genetic algorithm of Reeves [11]. Srikar and Ghosh [12] introduced the first MILP model (SG/SDST model) for the SDST flow-shop . Stafford and Tseng [13] reported minor corrections to the SG/SDST model, and showed that the revised model (SGST/SDST) was robust with regard to the triangular inequality relationship of setup times. Mercado and Bard [14] showed that the SGST/SDST model performed better than their new MILP model for the SDST flow-shop. Tseng and Stafford [15] extended the Stafford MILP model to solve both the SDST and the SDST/NIQ flow-shop problems. Mercado and Bard published two papers for the sequence dependent flow-shop problem with make-span criterion (denoted as $F_m/STsd/C_{max}$). In the first paper [16], they presented a branch and bound algorithm, incorporating lower and upper bounds and dominance elimination criterion, to solve the problem. They provided test results for a wide range of problem instances. In the second paper [17], they proposed a heuristic for the same problem, which transforms an instance of the problem into an instance of the traveling salesman problem by introducing a cost function that penalizes both large setup times and bad fitness of a given schedule. Ruiz et al. [18] proposed two heuristics for the same problem, and showed that their heuristics outperform that of Mercado and Bard [17] and others. Ruiz and Stutzle [19] presented two simple local search based iterated greedy algorithms, and showed that their algorithms perform better than those of Ruiz et al. [18]. Mercado and Bard [20] studied the polyhedral structure of two different mixed-integer programming formulations for the same problem. One is related to the asymmetric traveling salesman problem and the other is derived from an earlier proposed model. The two approaches were evaluated using a branch and cut algorithm, which indicated that the approach related to the asymmetric traveling salesman problem, was inferior in terms of the computational time. Stafford and Tseng [20] also proposed two mixed integer linear programming models, which are based on the work of Tseng and Stafford [18], for the same problem. The mixed-integer programming models proposed by Mercado and Bard [20] and Stafford and Tseng [21] were independently developed, and hence, remain to be compared to each other. Tseng et al. [22] developed a penalty-based heuristic algorithm for the same problem and compared their heuristic with an existing index heuristic algorithm. Sun and Hwang [23] addressed a related problem of $F_2/STsd/C_{max}$, where the setup times are present only on the second machine and the setup time of a job depends on *k* (*k*>1) immediately preceding jobs. They proposed a dynamic programming formulation and a genetic algorithm for the problem. Li et al. [24] presented partial enumeration method (PEM) to minimize the make-span performance of large flow-shop scheduling problems. The PEM run in short time and was able to easily combine with other algorithms or rules to improve performance. In their research two priority rules, variance method and variance-mean method were developed. Laha and Chakraborty [25] developed an efficient stochastic hybrid heuristic (H3) for flow-shop scheduling problem and showed the superiority of their work against other researches. Finally Sheibani [26] described a polynomial-time heuristic (PH) for the permutation flow-shop scheduling problem with the make-span criterion. His method consists of two phases: arranging the jobs in priority order and then constructing a sequence. He employed a fuzzy greedy evaluation function to prioritize the jobs for incorporating into the construction phase of the heuristic.

III. Hybrid genetic algorithm

Genetic algorithm (GA) developed by John Holland in the 1960s, is a stochastic optimization technique. Similar to other artificial intelligence heuristics like SA and TS, GA can avoid getting trapped in a local optimum by the aid of one of the genetic operations called mutation. The basic idea of GA is to maintain a population of candidate solutions that evolves under selective pressure. Hence, it can be viewed as a class of local search based on a solution generation mechanism operating on attributes of a set of solutions rather than attributes of a single solution by the move-generation mechanism of the local search methods, like SA and TS [27]. In recent years, GA has been applied successfully to a wide variety of hard optimization problems. The success is mainly due to its simplicity, easy operation, and great flexibility. These are the major reasons why GA is selected as an optimization tool in this paper. Flow-shop scheduling problem can be regarded as a hard optimization problem. A simple GA may not perform well in this situation. Therefore, the GA developed in this paper is hybridized with several heuristics to improve the solution further.Fig.1 shows the flowchart of HGA for the FSP. HGA hybridizes an improved heuristic called the iterated swap procedure (ISP). Besides the ISP, it also hybridizes the modified NEH_RMB proposed by Ruiz at al. [18] to generate a population of initial chromosomes. Also the author uses three genetic operators to make good new offspring.

The procedure of the HGA is described as follows: After the GA parameters, such as the iteration number, the population size, the crossover rate, and the mutation rate, have been set; the HGA generates the initial chromosomes of the problem. In HGA the NEH is used to determine the sequence of jobs in each schedule. After the predetermined number of initial chromosomes is generated, the ISP is adopted to improve all chromosomes.

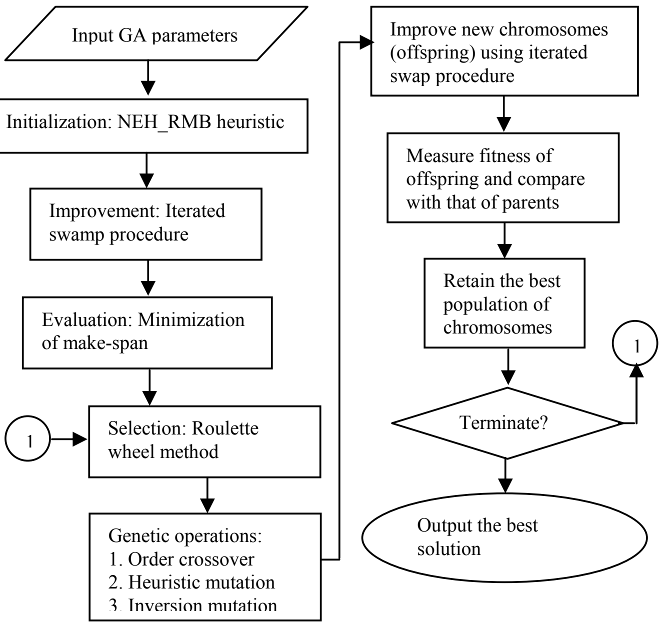
Input GA parameters
Initialization: NEH_RMB heuristic
Improvement: Iterated swamp procedure
Evaluation: Minimization of make-span
1
Selection: Roulette wheel method
Genetic operations:
1. Order crossover
2. Heuristic mutation
3. Inversion mutation
Improve new chromosomes (offspring) using iterated swap procedure
Measure fitness of offspring and compare with that of parents
Retain the best population of chromosomes
Terminate?
1
Output the best solution

Fig. 1 The flowchart of the HGA

Each chromosome is then measured by an evaluation function. The roulette wheel selection operation is adopted to select some chromosomes for the genetic operations, including the order crossover, the heuristic mutation, and the inversion mutation. After a new chromosome or offspring is produced, its links are improved by the ISP. The fitness of the offspring is measured and the offspring may become a member of the population if it possesses a relatively good quality. These steps form iteration, and then the roulette wheel selection is performed again to start the next iteration. The HGA will not stop unless the predetermined number of iterations is conducted.

A. *Initialization*

The initial solution for HGA is ideally generated by a high performance construction heuristic. For the sequence dependent flow-shop scheduling with make-span criterion, the author uses the NEHT_RMB heuristic and a modified NEHT_RMB heuristic proposed by Ruiz et al. [18] for the initialization of the population. Recall that NEH is an insertion heuristic, where at each step the next unscheduled job is tentatively inserted in each possible position of some partial solution. The job is then finally inserted into the position where the objective function takes the lowest value. For executing such an insertion heuristic, the jobs need to be ordered in some way. For more details how this is done in NEHT_RMB and modified NEHT_RMB, the reader is referred to Ruiz et al. [18].

B. *Improvement*

The 2-opt local search heuristic is generally used to improve the solutions of the hard optimization problems. However, it increases the computational time because every two swaps are examined. If a new solution generated is better than the original one, or parent, in terms of quality, it will replace and become the parent. All two swaps are examined again until there is no further improvement in the parent. To increase efficiency, the ISP [28, 29] shown in Fig. 2, is used to improve the links of each initial solution and each offspring generated by the three genetic operators. The principle of the ISP is similar to that of the 2-opt local search heuristic, except that some instead of all two swaps are examined. The procedure of the ISP is as follows:

Step 1: Select two genes randomly from a link of a parent.
Step 2: Exchange the positions of the two genes to form an offspring.
Step 3: Swap the neighbors of the two genes to form four more offspring.
Step 4: Evaluate all offspring and find the best one.
Step 5: If the best offspring is better than the parent, replace the parent with the best offspring and go back to Step 1; otherwise, stop.

Select two genes randomly

Parent	1	$\boxed{4}$	5	$\boxed{2}$	3
Offspring 1	1	2	5	4	3
Offspring 2	2	1	5	4	3
Offspring 3	1	5	2	4	3
Offspring 4	1	2	5	3	4
Offspring 5	1	2	4	5	3
Offspring 1	1	2	5	4	3

Fig. 2 The iterated swap procedure

C. *Evaluation*

As mentioned before, the fitness function is minimizing the maximum completion time across all jobs (also called make-span and denoted by C_{max}).

D. *Selection*

The roulette wheel selection operation [30] is adopted to choose some chromosomes to undergo genetic operations. The approach is based on an observation that a roulette wheel has a section allocated for each chromosome in the population, and the size of each section is proportional to the chromosome's fitness. The fitter the chromosome, the higher the probability of being selected. Although one chromosome has the highest fitness, there is no guarantee it will be selected. The only certain thing is that, on average, a chromosome will be chosen with the probability proportional to its fitness. Suppose the population size is P_{size}, then the selection procedure is as follows:

Step 1: Calculate the total fitness of the population:

$$F = \sum_{h=1}^{Psize} C_{\max}(h)$$

Step 2: Calculate the selection probability P_h for each chromosome X_h:

$$P_h = \frac{F - C_{\max}(h)}{F * (P_{size} - 1)} \qquad \text{h=1, 2, …, } P_{size}$$

Step 3: Calculate the cumulative probability Q_h for each chromosome X_h:

$$Q_h = \sum_{j=1}^{h} P_j \qquad \text{h=1, 2, …, } P_{size}$$

Step 4: Generate a random number r in the range (0, 1].

Step 5: If Q_h -1<r≤Q_h, then chromosome X_h is selected.

E. Genetic Operation

The genetic search progress is obtained by two essential genetic operations, including exploitation and exploration. Generally, the crossover operator exploits a better solution while the mutation operator explores a wider search space. The genetic operators used in the algorithms for the flow-shop problem are one crossover and two mutations, which are called the heuristic mutation and the inversion mutation, respectively.

E. 1. The Order Crossover

The crossover operator adopted in the HGA is the classical order crossover [31], and two offspring will be generated at each time. The procedure of the order crossover operation is:

Step 1: Select a substring from the first parent randomly.

Step 2: Produce a protochild by copying the substring into the corresponding positions in the protochild.

Step 3: Delete those genes in the substring from the second parent. The resulting genes form a sequence.

Step 4: Place the genes into the unfilled positions of the protochild from left to right according to the resulting sequence of genes in Step 3 to produce an offspring, shown in Fig. 3.

Step 5: Repeat Steps 1–4 to produce another offspring by exchanging the two parents.

E. 2. The heuristic mutation

A heuristic mutation [31] is designed with the neighborhood technique to produce a better offspring. A set of chromosomes transformed from a parent by exchanging some genes is regarded as the neighborhood. Only the best one in the neighborhood is used as the offspring produced by the mutation. However, the purpose of the mutation operation is to promote diversity of the population. Therefore, it is necessary to change the original heuristic mutation for the FSP. The modification is that all neighbors generated are used as the offspring. The procedure of the heuristic mutation operation, shown in Fig. 4, is taken as follows:

Step 1: Pick up three genes in a parent at random.

Step 2: Generate neighbors for all possible permutations of the selected genes, and all neighbors generated are regarded as the offspring.

E. 3. The Inversion Mutation

The inversion operator [31], shown in Fig. 5, selects a substring from a parent and flips it to form an offspring. However, the inversion operator works with one chromosome only. It is similar to the heuristic mutation and thus lacks the interchange of characteristics between chromosomes. So, the inversion operator is a mutation operation, which is used to increase the diversity of the population rather than to enhance the quality of the population.

		Selected substring			
Parent 1	1	2	5	4	3
Parent 2	1	4	3	2	5
Offspring 1	1	2	5	4	3

Fig. 3 The order crossover operator

	Select three genes randomly				
Parent	$\boxed{1}$	2	$\boxed{5}$	4	$\boxed{3}$
Offspring 1	1	2	3	4	5
Offspring 2	3	2	1	4	5
Offspring 3	3	2	5	4	1
Offspring 4	5	2	1	4	3
Offspring 5	5	2	3	4	1
Offspring 1	1	2	3	4	5

Fig. 4 The heuristic mutation operator

		Selected substring			
Parent 1	1	2	5	4	3
Offspring 1	1	4	5	2	3

Fig. 5 The inversion mutation operator

IV. Result analysis

In this section, a computational study is carried out to compare the HGA with three best recently developed heuristics. I mean PEM presented by Li et al. [24]; H3 developed by Laha and Chakraborty [25] and PH described by Sheibani [26]. Four methods are compared using different problem sizes (*n*=10, 20, 30, 40, 50, 100 and *m*=5, 10, 15, 20). For each class of the problem defined by given (*n*, *m*), 10 instances of problem are randomly generated. Thus we obtain a total of 280 problem instances. Processing time and setup time are given from Uniform random U(1, 99) and U(1, 9) discrete distributions respectively. The numerical results are averaged through each ten instances.

The parameters of the HGA for the problems are:

Population size=20, Crossover rate=0.5 and Mutation rate=0.2.

Therefore, five pairs of chromosome are selected to perform the order crossover operation, whereas four chromosomes perform the heuristic mutation operation and the inversion mutation operation. The total number of offspring produced per iteration will be 34 (10 from the order crossover operation, 20 from the heuristic mutation operation, and 4 from the inversion mutation operation). The platform of our experiments is a personal computer with a Pentium-III 1.2 Hz CPU and 512 MB RAM. The programs are coded in MATLAB. Also to have equal condition between four methods all algorithms are run 10 independent times with a stopping criterion based on an elapsed CPU time given by $n \times m / 10$ seconds. This allows for more time as the number of jobs *n* and the number of machines *m* grows.

The make-span values appearing in Table 1 are averages of 10 make-spans obtained from as many problem instances. The results indicate that HGA produces solutions of a better quality than other methods.

For evaluating the different algorithms, the author used the performance measure (PM) stated as:

$$PM = \frac{Heu_{sol} - Best_{sol}}{Best_{sol}} \tag{1}$$

where $\mathrm{Heu_{sol}}$ is the make-span obtained by a given algorithm and $\mathrm{Best_{sol}}$ is the make-span of the best solution obtained by all algorithms.

V. Conclusions

In this paper the author studies the flow-shop scheduling problem in sequence dependent condition to challenge a large number of real world problems. FSP is a hard optimization problem and the author develops one meta-heuristic approach based on genetic algorithm called HGA to solve it. Genetic algorithm hybridized with an improved heuristic called the iterated swap procedure (ISP). Besides the ISP, it also hybridized the modified NEH_RMB proposed by Ruiz at al. to generate a population of initial chromosomes. Also the author uses three genetic operators to make good new offspring. Computational results demonstrate the performance of our method compared to some of the strong methods recently developed. It is noticeable when we see the most differences between HGA and the best method among considered approaches are also significant in the level of $\alpha = 0.05$.

References

[1] B. Ekşioğlu, S.D. Ekşioğlu and P. Jain, "A tabu search algorithm for the flow-shop scheduling problem with changing neighborhoods", Computers & Industrial Engineering, Vol. 54, 2008, pp. 1-11.
[2] Allahverdi, C.T. Ng, T.C.E. Cheng and M.Y. Kovalyov, "A survey of scheduling problems with setup times or costs", European Journal of Operational Research, Vol. 187, 2008, pp. 985–1032.
[3] J.N.D. Gupta, "Flow-shop schedules with sequence dependent setup times" Journal of the Operations Research Society of Japan, Vol. 29, 1986, pp. 206–219.
[4] J.N.D. Gupta and W.P. Darrow, "The two-machine sequence dependent flow-shop scheduling problem", European Journal of Operational Research, Vol. 24, 1986, pp. 439–446.
[5] S.M. Johnson, "Optimal two- and three-stage production schedules with setup times included", Naval Research Logistics Quarterly, Vol. 1, 1954, pp. 61–68.
[6] M.R. Garey, D.S. Johnson and R. Sethi, "The complexity of flow-shop and jobshop scheduling", Mathematics of Operations Research, Vol. 1, 1976, pp. 117–129.
[7] H.G. Campbell, R.A. Dudek and M.L. Smith, "A heuristic algorithm for the n job, m machine sequencing problem", Management Science, Vol. 16, 1970, pp. B630–B637.
[8] M. Nawaz, E.E. Enscore and I. Ham, "A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem", OMEGA, The International Journal of Management Science, Vol. 11, 1983, pp. 91–95.
[9] I.H. Osman and C.N. Potts, "Simulated annealing for permutation flow-shop scheduling", OMEGA, The International Journal of Management Science, Vol. 17, 1989, pp. 551–557.
[10] M. Widmer and A. Hertz, "A new heuristic method for the flow shop sequencing problem", European Journal of Operational Research, Vo. 41, 1989, pp. 186–193.
[11] C.R. Reeves, "A genetic algorithm for flow-shop sequencing" Computers and Operations Research, Vol. 22, 1995, pp. 5–13.
[12] B.N. Srikar and S. Ghosh, "A MILP model for the n-job M-stage flow-shop with sequence dependent set-up times", International Journal of Production Research, Vo. 24, 1986, pp. 1459–1474.
[13] E.F. Stafford and T.F. Tseng, "On the Srikar–Ghosh MILP model for the N×M SDST flow-shop problem", International Journal of Production Research, Vol. 28, 1990, pp. 1817–1830.
[14] R.Z. Ríos-Mercado and J.F. Bard, "Computational experience with a branch-and-cut algorithm for flow-shop scheduling with setups", Computers & Operations Research, Vol. 25, 1998, pp. 351–366.
[15] F.T. Tseng and E.F. Stafford, "Two MILP models for the N×M SDST flow-shop sequencing problem", International Journal of Production Research, Vol. 39, 2001, pp. 1777–1809.
[16] R.Z. Ríos-Mercado and J.F. Bard, "A branch-and-bound algorithm for permutation flow shops with sequence-dependent setup times", IIE Transactions, Vol. 31, 1999a, pp. 721–731.
[17] R.Z. Ríos-Mercado and J.F. Bard, An enhanced TSP-based heuristic for makespan minimization in a flow shop with setup times, Journal of Heuristics, Vol. 5, 1999b, pp.53–70.
[18] R. Ruiz, C. Maroto and J. Alcaraz, "Solving the flow-shop scheduling problem with sequence dependent setup times using advanced metaheuristics", European Journal of Operational Research, Vol. 165, 2005, pp. 34–54.
[19] R. Ruiz and T. Stutzle, "An iterated greedy heuristic for the sequence dependent setup times flow-shop with makespan and weighted tardiness objectives", European Journal of Operational Research, Vol. 87, 2008, pp. 1143-1159.
[20] R.Z. Ríos-Mercado and J.F. Bard, "The flow shop scheduling polyhedron with setup times", Journal of Combinatorial Optimization, Vol. 7, 2003, pp. 291–318.
[21] E.F. Stafford and F.T. Tseng, "Two models for a family of flow-shop sequencing problems", European Journal of Operational Research, Vol. 142, 2002, pp. 282–293.
[22] F.T. Tseng, J.N.D. Gupta and E.F. Stafford, "A penalty-based heuristic algorithm for the permutation flow-shop scheduling problem with sequence-dependent set-up times", Journal of the Operational Research Society, Vol. 57, 2005, pp. 541–551.
[23] J.U. Sun, and H. Hwang, "Scheduling problem in a twomachine flow line with the N-step prior-job-dependent set-up times", International Journal of Systems Science, Vol. 32, 2001, pp. 375–385.
[24] X. Li, Y. Wang and C. Wu, "Heuristic algorithms for large flow-shop scheduling problems", Intelligent Control and Automation, Vol. 4, 2004, pp. 2999 – 3003.
[25] D. Laha and U.K. Chakraborty, "An efficient stochastic hybrid heuristic for flow-shop scheduling", Engineering Applications of Artificial Intelligence, Vol. 20, 2007, pp. 851–856.
[26] K. Sheibani, "A fuzzy greedy heuristic for permutation flow-shop scheduling" Journal of the Operational Research Society, Vol. 61, 2010, pp. 813-818.
[27] I.H. Osman and J.P. Kelly, Meta-heuristics: Theory and Applications. Kluwer Academic Publishers, Boston, 1996.
[28] W. Ho and P. Ji, "Component scheduling for chip shooter machines: a hybrid genetic algorithm approach", Computers and Operations Research, Vol. 30, 2003, pp. 2175–2189.
[29] W. Ho and P. Ji, "A hybrid genetic algorithm for component sequencing and feeder arrangement", Journal of Intelligent Manufacturing, Vol. 15, 2004, pp. 307–315.
[30] D.E. Goldberg, "Genetic Algorithms in Search, Optimization and Machine Learning", Addison-Wesley, New York, 1989.
[31] M. Gen and R. Cheng, "Genetic Algorithms and Engineering Design", Wiley, New York, 1997.

TABLE I
PM VALUES FOR COMPARISON STUDIES BETWEEN ALGORITHMS (TIMES ARE IN SECOND)

Class of problem	*n*	*m*	HGA			
			Min PM	Average		Max PM
				PM	Time	
1	10	5	0_6	0.075	1.05	0.241
2	10	10	0_7	0.037	1.21	0.199
3	10	15	0_5	0.215	2.39	0.581
4	10	20	0_6	0.154	5.11	0.765
5	20	5	0_7	0.050	1.55	0.470
6	20	10	0_6	0.179	1.76	0.673
7	20	15	0_9	0.029	3.86	0.093
8	20	20	0_5	0.114	6.87	0.325
9	30	5	0_5	0.118	1.77	0.673
10	30	10	0_7	0.131	3.22	0.550
11	30	15	0_5	0.160	4.48	0.301
12	30	20	0_9	0.010	8.00	0.081
13	40	5	0_8	0.028	2.20	0.216
14	40	10	0_6	0.267	2.86	0.886
15	40	15	0_7	0.098	4.37	0.390
16	40	20	0_4	0.104	9.41	0.321
17	50	5	0_6	0.072	2.92	0.329
18	50	10	0_{10}	0.000	5.47	0.00
19	50	15	0_5	0.260	8.94	0.795
20	50	20	0_5	0.285	11.15	0.767
21	100	5	0_9	0.037	4.50	0.258
22	100	10	0_7	0.047	9.64	0.335
23	100	15	0_5	0.168	17.38	0.459
24	100	20	0_4	0.084	31.27	0.221
25	200	5	0_6	0.094	11.22	0.218
26	200	10	0_7	0.068	32.54	0.286
27	200	15	0_7	0.130	43.36	0.516
28	200	20	0_9	0.019	78.78	0.103