ORIGINAL RESEARCH

A novel hybrid genetic algorithm to solve the make-to-order sequence-dependent flow-shop scheduling problem

Mohammad Mirabi · S. M. T. Fatemi Ghomi · F. Jolai

Abstract Flow-shop scheduling problem (FSP) deals with the scheduling of a set of n jobs that visit a set of m machines in the same order. As the FSP is NP-hard, there is no efficient algorithm to reach the optimal solution of the problem. To minimize the holding, delay and setup costs of large permutation flow-shop scheduling problems with sequence-dependent setup times on each machine, this paper develops a novel hybrid genetic algorithm (HGA) with three genetic operators. Proposed HGA applies a modified approach to generate a pool of initial solutions, and also uses an improved heuristic called the iterated swap procedure to improve the initial solutions. We consider the make-to-order production approach that some sequences between jobs are assumed as tabu based on maximum allowable setup cost. In addition, 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

M. Mirabi (✉)
Group of Industrial Engineering, Ayatollah Haeri University of Meybod, P.O. Box 89619-55133, Meybod, Iran
e-mail: M.Mirabi@yahoo.com

S. M. T. Fatemi Ghomi
Department of Industrial Engineering, Amirkabir University of Technology, P.O. Box 15916-34311, Tehran, Iran
e-mail: Fatemi@aut.ac.ir

F. Jolai
Department of Industrial Engineering, College of Engineering, University of Tehran, P.O. Box 14395-515, Tehran, Iran
e-mail: Fjolai@ut.ac.ir

Introduction

It is almost seven decades that flow-shop scheduling problems (FSP) have been studied as a major field of study in manufacturing researches. In an m machine flow shop, there are m stages in series with one or more machines at each. Also, there are n jobs that each one has to be processed in each of the m stages in the same order. In the classical flow-shop problem, there is one machine at each stage and this field attracts the most attendances. Two major sub-problems of FSP are sequence-independent setup time (SIST) and sequence-dependent setup time (SDST). The SDST flow-shop problem is more compatible with the real-world problem, but has attracted much less attention, especially before 2000 (Allahverdi et al. 2008).

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 $(n!) \times m$ possible processing sequences. Flow-shop scheduling researches usually only attend permutation sequences where the processing order of operations is the same for all machines. Here, we also adopt this restriction.

Minimizing the maximum completion time across all jobs (also called make-span and denoted by C_{max}) is the most well-known and applicable criterion in the literature. Regarding the computational complexity, the SDST flow-shop with the C_{max} objective has been shown to be NP-hard by Gupta and Darrow (1986), even when $m = 1$ and also when $m = 2$ and setups are presented only on the first or second machine. Therefore, solving the problem by an exact algorithm is time consuming and computationally intractable.

Pioneering work was due to Johnson (1954) who proposed a simple rule to obtain optimal sequences for the

permutation flow-shop problem (PFSP) with two machines. This work was the starting point of several attempts for solving the PFSP with more than two machines. Because of NP completeness of the PFSP (Garey et al. 1976; Campbell et al. 1970), researchers have mainly attempted to the development of effective heuristics and meta-heuristics. Some of the first heuristic methods in this field are the well-known NEH heuristic by Nawaz et al. (1983) and the genetic algorithm of Reeves (1995).

After 2000, there are wide range of heuristics, meta-heuristics and hybrid meta-heuristics developed for flow-shop and permutation flow-shop by researchers. Ruiz et al. (2005) proposed two heuristics for the same problem, and showed that their heuristics outperform previous ones. Ruiz and Stutzle (2008) presented two simple local search-based iterated greedy algorithms, and showed that their algorithms perform better than those of Ruiz et al. (2005). Tseng et al. (2005) developed a penalty-based heuristic algorithm for the same problem and compared their heuristic with an existing index heuristic algorithm. Among all approaches, one of the most successful meta-heuristics to solve PFSP from last until now is genetic algorithm. Like one worked by Reeves (1995) and Sun and Hwang (2001). Sun and Hwang (2001) addressed a related problem of $F_2/ST_{sd}/C_{max}$ 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. Chaari et al. (2011) considered a scheduling problem under uncertainty. They developed a genetic algorithm for the case of hybrid flow-shop scheduling problem that the processing time of each job for each machine at each stage is the source of uncertainty. They defined a robust bi-objective evaluation function to obtain a robust, effective solution that is only slightly sensitive to data uncertainty. Tseng and Lin (2010) proposed a hybrid genetic algorithm to solve the no-wait flow-shop scheduling problem with the make-span objective. The proposed algorithm hybridized the genetic algorithm and a novel local search scheme. The proposed local search scheme combines two local search methods: the insertion search and a novel local search method called the insertion search with cut-and-repair. Jarboui et al. (2011) proposed a hybrid genetic algorithm to minimize the make-span and the total flow time in the no-wait flow-shop scheduling problem. In their research, the variable neighborhood search was used as an improvement procedure in the last step of the genetic algorithm. Huang and Huang (2010) considered a flow-shop scheduling problem with synchronous material movement in an automated machine center consisting of a loading/unloading (L/U) station, m processing machines, and a rotary table. Furthermore, other useful and strong approaches can be fined to solve PFSP. Li et al. (2004)

presented partial enumeration method (PEM) to minimize the make-span performance of large flow-shop scheduling problems. The PEM run in short time and could 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 (2007) developed an efficient stochastic hybrid heuristic (H3) for flow-shop scheduling problem and showed the superiority of their work against other researches. Noori-Darvish and Tavakkoli-Moghaddam (2012) proposed a novel bi-objective mathematical programming for an open-shop scheduling problem with setup and processing times separately such that not only the setup times are dependent on the machines, but also they are dependent on the sequence of jobs that should be processed on a machine. They minimized the total tardiness and the make-span. Maleki-Darounkolaei et al. (2012) considered a three-stage assembly flow-shop scheduling problem with sequence-dependent setup times at the first stage and blocking times between each stage in such a way that the weighted mean completion time and make-span are minimized. Finally, Sheibani (2010) 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.

Successful applications of GA to solve NP-hard problems such as FSP stimulated us to develop one hybrid GA (HGA) to deal with the problem efficiently and effectively.

As mentioned before in the classical flow-shop problem, the make-span minimization criterion has always attracted the attention of researchers. With a fast glance to the real-world situations, we can see that due date and setup costs are the most important criteria in production planning, especially in the make-to-order situation. Various customers offered their orders (jobs) and each order has its own due date, holding cost and delay cost, and just focus on the make-span is not an effective attempt. Almost all real-world problems are multi-criteria and considering just one criterion is too far from real situations (Mirabi 2010). Allahverdi et al. (2008) implied that almost no multi-criteria researches are available according to real situations. He also suggested the due date-related criteria for more consideration.

As mentioned before, SDST is more adaptable with real situations. Furthermore, in wide cases there are some infeasible sequences (Tabu sequences) based on setup cost. For example, in the dyeing process of all kinds of fibers, each basket filled by wet fiber (job) must be dyed by several dyeing machines (stages). Dyeing completely the bright color (cream, white) after completely the dark color

(black, blue and red) caused high setup costs (after dark color, each machine must be cleaned carefully for almost 2 days and also at least one batch of bright color waste for the reason of remain dyes from dark color) and they are considered as infeasible sequences. These are the same conditions used in the cable industry for producing colored wires.

In this research, we attend the permutation flow shop that must process n jobs, each one received from specific customer. Each job has specific due date and delay cost. Setup time and setup cost are sequence dependent and also objective function constructs by three criteria as delay, holding and setup costs. Also, some sequences known as Tabu based on yielded setup costs. This situation is compatible with the large category of the real-world problems, and is completely missed in the literature. We develop one HGA to solve the problem.

The paper is organized as follows: Section 2 discusses the principles of the algorithms used to solve the permutation FSP. Section 3 compares the performance of the algorithms. Finally, Sect. 4 concludes the paper.

Hybrid genetic algorithm

In this research, genetic algorithm (GA) is applied to solve permutation flow-shop scheduling problem. John Holland proposed GA in the 1960s for the first time. GA categorizes as a class of evolutionary algorithms (EA), which generate solutions to optimization and heuristic problems by techniques inspired by natural evolution, such as mutation, selection, and crossover. The main concept of GA is to evolve a population of candidate solutions (called individuals, creatures, or phenotypes) to an optimization problem toward better solutions. Today, GA has wide and successful applications to solve hard optimization problems. The success is mainly due to its easy to understand and operation and great flexibility. These reasons stimulate us to use this strong approach to solve the presented problem.

Initially, many individual solutions (called chromosomes) are (usually) randomly generated to generate an initial population. The population size depends on the nature of the problem, but generally contains several hundreds or thousands of possible solutions. The chromosomes evolve through successive iterations, called generations. During each generation, the chromosomes are evaluated through a fitness-based process where fitter solutions (as measured by a fitness function) are typically more likely to be selected. The next step is to generate a second-generation population of solutions from those selected through genetic operators: crossover and mutation.

For each new solution to be produced, a pair of "parent" solutions is selected for breeding from the pool selected previously. By producing a "child" solution using the above methods of crossover and mutation, a new solution is created which typically shares many of the characteristics of its "parents". New parents are selected for each new child, and the process continues until a new population of solutions of appropriate size is generated.

After a fixed number of generations, the algorithm converges to the best chromosome, which probably is the optimal solution or may be a near-optimal solution of the problem.

Flow-shop scheduling problem can be regarded as a hard optimization problem and to enrich the capability of the proposed GA in this paper it is hybridized with some other approaches. The GA developed in this paper is hybridized with several heuristics to improve the solution further.

Figure 1 shows the flowchart of HGA for the FSP. HGA hybridized with an improved heuristic called the iterated swap procedure (ISP). Besides the ISP, it also hybridized the heuristic method to construct a pool of initial solutions. Also, the author uses three genetic operators to make a good new offspring.

The procedure of the HGA is described as follows: After the GA parameters, such as the iteration number, the population size (P_{size}), the crossover rate, and the mutation rate, have been set, the HGA generates the initial chromosomes of the problem. After the predetermined number of initial chromosomes is generated, the ISP is adopted to improve all chromosomes. 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.

Initialization

The initial solution for HGA is ideally generated by a high performance construction heuristic. In the initialization phase, we need one pool of initial solutions and based on our experience, construction heuristic works better that random approach. Before all, we construct a list of Tabu sequences based on earned information about setup costs. After that, we classify all orders (jobs) in four levels based

Fig. 1   The flowchart of the HGA
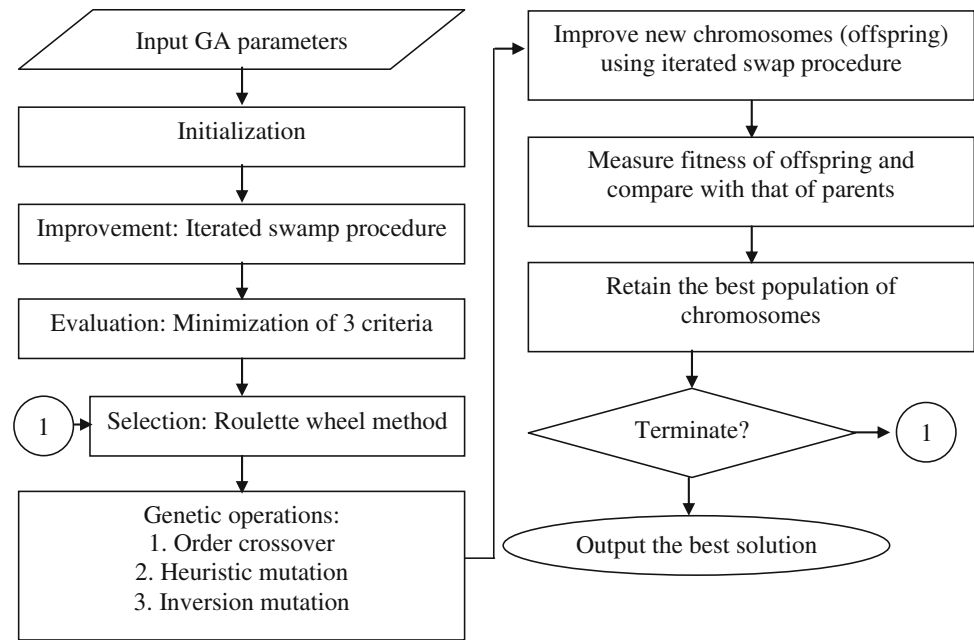
| 8 | 4 | 0 | 8 | 16 | 24 | 32 | 40 | 48 | 56 |

Fig. 2   String related to job 1

on their importance. Importance of each job is determined by its customer importance and ranking (based on ISO standard list), job delay cost, job holding cost and other factors based on the management viewpoint. Levels names are low, medium, high and very high. Jobs in very high level have the most importance and jobs in low level have the least.

Each job has its own due date and by delivering before and after due date, holding and delay cost per unit of period must be paid, respectively. For example, assumed delivery time of job numbered 1 is period 4, and the process time is two periods. Also assumed holding cost and delay cost per period is 4 and 8, and planning horizon has 10 time periods. For job 1, we can construct a string like Fig. 2.

It means produce job 1 in period 1 costs 8 and so on. For n jobs we have n strings like this. For initialization the following algorithm is used:

1) Input T (number of time period) n (number of Job) and P_{size} (population size)
2) Numbered jobs from 1 to n and time periods from 1 to T
3) For $t = 1$ to T
4) $t = 1$
5) Select a job with the least cost in period t and put it as soon as possible in the list of sequence. If two jobs have the same cost, select a job with more importance and with the same conditions select one in random

6) Eliminate the string related to selected job
7) If there is free capacity in period t go to 4
8) If not $t = t + 1$
9) End for
10) Extract final sequence (initial solution 1)
11) For $k = 1$ to P_{size}
12) Interchange the position of two jobs in the same level (by random)
13) If there is no Tabu sequence, finalize the result as one initial solution
14) End for
15) Extract all P_{size} initial solutions

Of course in the initialization phase, we do not consider the setup cost and just care about Tabu sequences.

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 (Ho and Ji 2003, 2004) shown in Fig. 3, 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:

	Select two genes randomly			
Parent	1	4	5	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. 3 The iterated swap procedure

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.

Evaluation

As mentioned before, the fitness function is minimizing the tardiness, holding, and setup costs (Mirabi 2010). For example, consider one solution from pool of solutions (called solution h). Fitness function of this selected solution is:

$$Z(h) = \sum_{j=0}^{n} \sum_{i=1}^{n} SC_{ij} \left(\sum_{j=0}^{n} X_{jit} \right)$$
$$+ \sum_{i=1}^{n} \sum_{t=DD_i+1}^{n} \left(\sum_{j=0}^{n} X_{jit} \times (t - DD_i) \times DC_i \right)$$
$$+ \sum_{j=1}^{n} \sum_{t=\min\{RT_i, DD_i\}}^{DD_i} \left(\sum_{j=0}^{n} X_{jit} \times (DD_i - t) \times HC_i \right)$$

where indices i and $j = 1,\dots,n$ index set of all jobs, $t = 1,\dots,T$ index set of all periods. All periods are assumed to be of equal length; parameters: PT_i is processing time of job i, DD_i due date of job i, ST_{0i} initial setup time of job i (when job i is the first job in sequence), ST_{ji} is setup time of job i when it is processed after job j, SC_{0i} is initial setup cost of job i (when job i is the first job in sequence), SC_{ji} is setup cost of job i when it process after job j, HC_i is holding cost of job i per each time period, DC_i is delay cost of job i per each time period, RT_i is release time of job i; variables: X_{jit} 1, if sequence ji (job i is processed after job j) appears in period t; 0, otherwise; X_{0i1} 1, if job i is the first job in sequence (obviously it produce in period 1); 0, otherwise.

The first term of the fitness function calculates the sum of all setup costs for each ji (ij) sequence in the production line. Second term shows that if each job i is produced after due date (), for each delay period, the penalty in accordance to DC_i should be paid. Finally, the last term mention producing after due date impose the delay cost.

In this research, we just work with the fitness function, but there are some constraints for the problem that we refer readers to Mirabi (2010).

Selection

The commonly used genetic operator is the roulette wheel selection operation (Goldberg 1989). It is the proportionate reproduction operator where a string is selected for the mating pool with a probability proportional to its fitness. Thus, the ith string in the population is selected with a probability proportional. The fitter is the chromosome, the higher is the probability of being selected. Although one chromosome has the highest fitness, there is no guarantee it will be selected. Since the population size is usually kept fixed in a simple GA, the sum of the probability of each string being selected for the mating pools must be 1. Suppose the population size is P_{size}, then the selection procedure is as follows:

Step 1: Calculate the total fitness of the population:

$$F = \sum_{h=1}^{P_{size}} Z(h)$$

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

$$P_h = \frac{F - Z(h)}{F * (P_{size} - 1)} \quad h = 1, 2, \dots, P_{size}$$

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

$$Q_h = \sum_{j=1}^{h} P_j \quad h = 1, 2, \dots, P_{size}$$

Step 4: Generate a random number r in the range (0, 1].
Step 5: If $Q_{h-1} < r \leq Q_h$, then chromosome X_h is selected.

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

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

Fig. 4 The order crossover operator

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

Fig. 6 The inversion mutation operator

		Select three genes randomly		
Parent	1	2	5	4	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. 5 The heuristic mutation operator

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.

The order crossover

The crossover operator adopted in the HGA is the classical order crossover (Gen and Cheng 1997), and two offsprings 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 an offspring by copying the substring into the corresponding positions in the offspring.
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 offspring from left to right according to the resulting sequence of genes in Step 3 to produce an offspring, shown in Fig. 4.
Step 5: Repeat Steps 1–4 to produce another offspring by exchanging the two parents.

The heuristic mutation

A heuristic mutation (Gen and Cheng 1997) 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. 5, 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.

The inversion mutation

The inversion operator (Gen and Cheng 1997), shown in Fig. 6, 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. Therefore, 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.

Result analysis

In this section, a computational study is carried out to compare the HGA with three best recently developed heuristics. We mean PEM presented by Li et al. (2004); H3 developed by Laha and Chakraborty (2007) and PH described by Sheibani (2010). 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), ten 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

Table 1 PM values for comparison studies between algorithms (times are in second)

Class of problem	n	m	HGA Min PM	HGA Average PM	HGA Time	HGA Max PM	PH Min PM	PH Average PM	PH Time	PH Max PM	H3 Min PM	H3 Average PM	H3 Time	H3 Max PM	PEM Min PM	PEM Average PM	PEM Time	PEM Max PM
1	10	5	0_6	0.075	1.05	0.241	0_3	0.217	1.83	0.809	0	0.422	0.36	1.026	0	0.679	0.26	1.117
2	10	10	0_7	0.037	1.21	0.199	0_2	0.436	1.84	1.087	0	0.528	6.77	1.017	0.135	0.878	0.66	2.070
3	10	15	0_5	0.215	2.39	0.581	0_4	0.387	2.78	1.050	0_2	0.735	28.53	1.367	0.362	1.306	0.62	2.273
4	10	20	0_6	0.154	5.11	0.765	0_4	0.225	4.01	0.996	0.005	0.839	88.28	1.534	0.003	0.944	0.71	2.172
5	20	5	0_7	0.050	1.55	0.470	0_3	0.425	2.15	1.064	0.309	0.841	0.44	1.454	0.026	1.203	0.37	2.016
6	20	10	0_6	0.179	1.76	0.673	0_2	0.536	2.77	1.239	0_2	0.740	6.79	1.559	0.305	1.201	0.60	2.201
7	20	15	0_9	0.029	3.86	0.093	0	0.574	2.94	0.981	0.036	0.614	34.56	1.114	0.095	0.882	0.62	1.505
8	20	20	0_5	0.114	6.87	0.325	0_5	0.169	4.05	0.575	0_1	0.480	87.31	1.056	0.100	0.647	0.72	1.344
9	30	5	0_5	0.118	1.77	0.673	0_5	0.131	2.76	0.629	0.028	0.587	0.43	1.136	0.079	0.826	0.46	1.495
10	30	10	0_7	0.131	3.22	0.550	0_3	0.213	3.59	0.774	0_2	0.384	8.85	1.099	0.375	1.003	0.78	1.672
11	30	15	0_5	0.160	4.48	0.301	0_5	0.130	4.34	0.397	0.118	0.486	47.25	0.967	0.279	0.749	0.82	1.119
12	30	20	0_9	0.010	8.00	0.081	0_1	0.390	6.10	0.816	0_1	0.648	129.53	1.117	0.023	0.822	1.10	1.584
13	40	5	0_8	0.028	2.20	0.216	0_2	0.381	3.24	0.817	0.296	0.797	0.40	1.227	0.053	0.621	0.56	1.178
14	40	10	0_6	0.267	2.86	0.886	0_4	0.233	4.29	0.654	0_1	0.381	10.62	0.776	0.311	0.817	0.90	1.436
15	40	15	0_7	0.098	4.37	0.390	0_3	0.340	5.61	0.810	0.144	0.412	58.11	0.974	0.033	0.528	1.03	1.474
16	40	20	0_4	0.104	9.41	0.321	0_6	0.375	7.83	0.718	0.055	0.430	149.62	0.884	0.227	0.659	1.49	1.046
17	50	5	0_6	0.072	2.92	0.329	0_4	0.146	3.87	0.536	0_1	0.416	0.40	1.010	0.001	0.569	0.71	1.237
18	50	10	0_{10}	0.000	5.47	0.00	0.005	0.417	5.11	0.723	0_1	0.456	12.63	0.895	0.082	0.388	1.07	0.994
19	50	15	0_5	0.260	8.94	0.795	0_5	0.235	7.04	0.672	0_1	0.454	70.84	0.980	0.121	0.799	1.26	1.336
20	50	20	0_5	0.285	11.15	0.767	0_4	0.242	9.89	0.695	0_1	0.472	200.86	0.895	0.068	0.579	1.90	0.937
21	100	5	0_9	0.037	4.50	0.258	0_1	0.317	6.14	0.488	0.023	0.299	0.96	0.673	0.011	0.383	0.98	0.719
22	100	10	0_7	0.047	9.64	0.335	0_3	0.213	13.22	0.542	0.009	0.370	19.22	0.769	0.214	0.470	2.84	0.795
23	100	15	0_5	0.168	17.38	0.459	0_5	0.202	18.59	0.492	0.037	0.236	109.94	0.676	0.031	0.387	4.03	0.833
24	100	20	0_4	0.084	31.27	0.221	0_4	0.217	26.49	0.458	0_2	0.377	345.28	0.595	0.120	0.480	6.74	0.898
25	200	5	0_6	0.094	11.22	0.218	0_4	0.520	14.55	0.867	0.012	0.386	4.66	0.874	0.032	0.463	5.61	1.099
26	200	10	0_7	0.068	32.54	0.286	0_3	0.252	26.79	0.754	0.329	0.663	27.74	0.997	0.050	0.561	7.06	1.107
27	200	15	0_7	0.130	43.36	0.516	0_3	0.257	32.15	0.663	0_1	0.448	221.28	0.946	0.220	0.792	14.01	1.675
28	200	20	0_9	0.019	78.78	0.103	0_1	0.361	69.32	0.766	0.068	0.409	700.60	0.808	0.053	0.740	32.13	1.223

Table 2 Detailed comparison between HGA and PH

Class of problem	n	M	Ave. MS or $(\overline{X})$		Ave. SD or (S)		T	υ	t	Sig.
			HGA	PH	HGA	PH				
1	10	5	759.38	766.46	2.88	2.55	5.81	18	1.73	Yes
2	10	10	1,105.13	1,111.11	2.53	1.28	6.66	13	1.77	Yes
3	10	15	1,287.82	1,290.31	2.87	4.18	1.56	16	1.75	No
4	10	20	1,605.76	1,615.15	4.61	3.65	5.05	17	1.74	Yes
5	20	5	1,326.39	1,331.05	3.2	3.31	3.20	18	1.73	Yes
6	20	10	1,584.28	1,590.6	4.56	5.5	2.80	17	1.74	Yes
7	20	15	1,875.41	1,879.35	2.62	2.34	3.55	18	1.73	Yes
8	20	20	2,150.99	2,144.26	4.53	4.16	−3.46	18	1.73	Yes
9	30	5	1,856.52	1,853.47	5.16	2.81	−1.64	14	1.76	No
10	30	10	2,160	2,163.15	4.84	4.36	1.52	18	1.73	No
11	30	15	2,453.33	2,448.04	4.58	1.79	−3.40	12	1.78	Yes
12	30	20	2,705.48	2,710.08	5.48	4.93	1.97	18	1.73	Yes
13	40	5	2,431.57	2,434.38	5.07	4.44	1.32	18	1.73	Yes
14	40	10	2,686.62	2,692.25	3.76	7.13	2.21	14	1.76	Yes
15	40	15	2,971.97	2,975.92	3.58	2.39	2.90	16	1.75	Yes
16	40	20	3,250.92	3,244.74	6.96	3.71	−2.48	14	1.76	Yes
17	50	5	2,958.28	2,963.57	6.12	5.94	1.96	18	1.73	Yes
18	50	10	3,258.9	3,261.99	7.53	5.96	1.02	17	1.74	Yes
19	50	15	3,519.04	3,511.88	8.42	7.19	−2.05	18	1.73	Yes
20	50	20	3,790.77	3,803.47	7.83	6.75	3.89	18	1.73	Yes
21	100	5	5,734.8	5,741.41	7.94	5.08	2.22	15	1.75	Yes
22	100	10	6,008.96	6,015.08	8.25	6.21	1.87	17	1.74	Yes
23	100	15	6,251.23	6,261.23	11.29	12.29	1.90	18	1.73	Yes
24	100	20	6,613.9	6,602.89	11.45	9.58	−2.33	17	1.74	Yes
25	200	5	11,229.39	11,241.2	14.29	14.16	1.86	18	1.73	Yes
26	200	10	11,651.74	11,657.97	23.74	22.78	0.60	18	1.73	No
27	200	15	11,846.48	11,847.81	19.51	22.93	0.14	18	1.73	No
28	200	20	12,297.14	12,306.81	22.54	21.5	0.98	18	1.73	No

Ave. average, *MS* make-span, *SD* standard deviation, *Sig.* significant

Each class contains ten independent instances

crossover operation, 20 from the heuristic mutation operation, and 4 from the inversion mutation operation). The platform of the 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 by the same iteration.

For evaluating the different algorithms, we used the performance measure (PM) for each class of problem stated as:

$$PM_i = \frac{Solution_i - Best_{sol}}{Best_{sol}} \qquad (1)$$

where Solution$_i$ is the fitness value obtained by algorithm i, and Best$_{sol}$ is the best fitness value between all algorithms.

The average, minimum, and maximum PM values for all algorithms are shown in Table 1. The columns labeled "Min" show, in subscript, the number of instances (between ten instances) for which the algorithm solution

was equal to the corresponding Best$_{sol}$. In the "average" column, we showed two subcolumns including the average of ten PM values related to ten instances and also the average of ten solution times to reach all ten results. With respect to the solutions gained, Table 1 demonstrates that the algorithms have the rank of 1 for HGA, 2 for PH, 3 for H3, and 4 for PEM.

Now, for more detailed comparison, two algorithms of HGA and PH are considered. In this step, it is desired to stop both algorithms at the same CPU time. The value of this CPU time has taken the minimum CPU time between the two algorithms in Table 1. For example, the common CPU time for the first class of problem ($n = 10$, $m = 5$) is min (1.05, 1.83) = 1.05.

The author now tests the hypothesis that the population corresponding to the differences has mean μ zero; specifically, test the (null) hypothesis μ against the alternative $\mu > 0$. It is assumed that the make-span difference is a normal variable, and choose the significance level

$\alpha = 0.05$. If the hypothesis is true, the random variable $T = (\overline{X}_1 - \overline{X}_2)/\sqrt{(S_1^2/n_1) + (S_2^2/n_2)}$ has a t distribution with: $\upsilon = (S_1^2/n_1 + S_2^2/n_2)^2/(\frac{(S_1^2/n_1)^2}{n_1-1} + \frac{(S_2^2/n_2)^2}{n_2-1})$ degrees of freedom (Mirabi 2010). The critical value of c is obtained from the relation $\text{Prob}(T > c) = \alpha = 0.05$. For example, the first entry in Table 1 corresponds to the sample size $= n_1 = n_2 = 10$, $\mu_0 = 0$, sample means for HGA and H3 are $\overline{X}_1 = 759.38$ and $\overline{X}_2 = 766.46$, respectively. Sample standard deviations for HGA and H3 are $S_1 = 2.88$ and $S_2 = 2.55$, respectively. Since $t = 1.73 < T = 5.81$, we conclude that the difference is statistically significant. Table 2 displays that HGA outperforms H3 in all class of problems except six (76 % superiority). More than 77 % of these superiorities are statistically significant.

Conclusions

In this paper, we studied the permutation flow-shop scheduling problem in sequence-dependent condition to challenge a large number of real-world problems in make-to-order production strategy. FSP is a hard optimization problem, and we develop 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 hybridized the heuristic approach to construct a pool of initial solutions. Also, we use three genetic operators to make a good new offspring. Computational results demonstrate the performance of presented 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

Allahverdi A, Ng CT, Cheng TCE, Kovalyov MY (2008) A survey of scheduling problems with setup times or costs. Eur J Oper Res 187:985–1032

Campbell HG, Dudek RA, Smith ML (1970) A heuristic algorithm for the n job, m machine sequencing problem. Manage Sci 16:B630–B637

Chaari T, Chaabane S, Loukil T, Loukil D (2011) A genetic algorithm for robust hybrid flow shop scheduling. Int J Comput Integr Manuf 24:821–833

Garey MR, Johnson DS, Sethi R (1976) The complexity of flow-shop and job-shop scheduling. Math Oper Res 1:117–129

Gen M, Cheng R (1997) Genetic algorithms and engineering design. Wiley, New York

Goldberg DE (1989) Genetic algorithms in search, optimization and machine learning. Addison-Wesley, New York

Gupta JND, Darrow WP (1986) The two-machine sequence dependent flow-shop scheduling problem. Eur J Oper Res 24:439–446

Ho W, Ji P (2003) Component scheduling for chip shooter machines: a hybrid genetic algorithm approach. Comput Oper Res 30:2175–2189

Ho W, Ji P (2004) A hybrid genetic algorithm for component sequencing and feeder arrangement. J Intell Manuf 15:307–315

Huang K (2010) Hybrid genetic algorithms for flow-shop scheduling with synchronous material movement. Computer and industrial engineering (CIE), 40th international conference, pp 1–6

Jarboui B, Eddaly M, Siarry P (2011) A hybrid genetic algorithm for solving no-wait flow-shop scheduling problems. Int J Adv Manuf Technol 54:1129–1143

Johnson SM (1954) Optimal two- and three-stage production schedules with setup times included. Naval Res Log Q 1:61–68

Laha D, Chakraborty UK (2007) An efficient stochastic hybrid heuristic for flow-shop scheduling. Eng Appl Artif Intell 20:851–856

Li X, Wang Y, Wu C (2004) Heuristic algorithms for large flow-shop scheduling problems. Intell Control Autom 4:2999–3003

Maleki-Darounkolaei A, Modiri M, Tavakkoli-Moghaddam R, Seyyedi I (2012) A three-stage assembly flow shop scheduling problem with blocking and sequence dependent set up times. J Ind Eng Int 8:26. http://www.jiei-tsb.com/content/8/1/26

Mirabi M (2010) A hybrid simulated annealing for the single machine capacitated lot-sizing and scheduling problem with sequence-dependent setup times and costs and dynamic release of jobs. Int J Adv Manuf Technol. doi:10.1007/s00170-010-2988-5

Nawaz M, Enscore EE, Ham I (1983) A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. OMEGA Int J Manage Sci 11:91–95

Noori-Darvish S, Tavakkoli-Moghaddam R (2012) Minimizing the total tardiness and make span in an open shop scheduling problem with sequence-dependent setup times. J Ind Eng Int 8:25. http://www.jiei-tsb.com/content/8/1/25

Reeves CR (1995) A genetic algorithm for flow-shop sequencing. Comput Oper Res 22:5–13

Ruiz R, Stutzle T (2008) An iterated greedy heuristic for the sequence dependent setup times flow-shop with make-span and weighted tardiness objectives. Eur J Oper Res 87:1143–1159

Ruiz R, Maroto C, Alcaraz J (2005) Solving the flow-shop scheduling problem with sequence dependent setup times using advanced meta-heuristics. Eur J Oper Res 165:34–54

Sheibani K (2010) A fuzzy greedy heuristic for permutation flow-shop scheduling. J Oper Res Soc 61:813–818

Sun JU, Hwang H (2001) Scheduling problem in a two machine flow line with the N-step prior-job-dependent set-up times. Int J Syst Sci 32:375–385

Tseng L, Lin Y (2010) A hybrid genetic algorithm for no-wait flow-shop scheduling problem. Int J Prod Econ 128:144–152

Tseng FT, Gupta JND, Stafford EF (2005) A penalty-based heuristic algorithm for the permutation flow-shop scheduling problem with sequence-dependent set-up times. J Oper Res Soc 57:541–551