Int. J. Production Economics 127 (2010) 121–128

Contents lists available at ScienceDirect

Int. J. Production Economics

journal homepage: www.elsevier.com/locate/ijpe

A genetic local search algorithm for minimizing total flowtime in the permutation flowshop scheduling problem

Lin-Yu Tseng [a,b,*], Ya-Tai Lin [b]

[a] Institute of Networking and Multimedia, National Chung Hsing University, 250 Kuo Kuang Road, Taichung, Taiwan
[b] Department of Computer Science and Engineering, National Chung Hsing University, 250 Kuo Kuang Road, Taichung, Taiwan

ARTICLE INFO

Article history:
Received 16 April 2007
Accepted 23 April 2010
Available online 1 June 2010

Keywords:
Genetic algorithm
Tabu search
Flowshop scheduling
Total flowtime
Genetic local search

ABSTRACT

Recently, the flowshop scheduling problem to minimize total flowtime has attracted more attention from researchers. In this paper, a genetic local search algorithm is proposed to solve this problem. The proposed algorithm hybridizes the genetic algorithm and the tabu search. It employs the genetic algorithm to do the global search and the tabu search to do the local search. The orthogonal-array-based crossover is utilized to enhance the capability of intensification. Also, a novel orthogonal-array-based mutation is proposed, in order to add capability of intensification to the traditional mutation operator. The performance of the proposed genetic local search algorithm is very competitive. It improved 54 out of 90 current best solutions reported in the literature for short-term search, and it also improved 18 out of 20 current best solutions reported in the literature for long-term search.

1. Introduction

The flowshop scheduling problem proposed by Johnson (Johnson, 1954) is an important scheduling problem, and has been extensively studied. Most of the research on this problem was to minimize makespan. In recent years, the flowshop scheduling problem to minimize total flowtime has started to attract more attention from researchers. The flowshop scheduling problem with this criterion is considered to be more relevant and meaningful for today's dynamic production environment (Liu and Reeves, 2001) because it tends to stabilize use of resources and minimize the work-in-process inventory. This problem is formally defined in the following.

n jobs $\{J_1, J_2, \ldots, J_n\}$ are to be processed on a series of m machines $\{M_1, M_2, \ldots, M_m\}$ sequentially. The processing time of job J_i on machine M_j is given as T_{ij}. At any time, each job can be processed on at most one machine and each machine can process at most one job. Also, once a job is processed on a machine, it cannot be terminated before completion. The sequence in which the jobs are to be processed is the same for each machine. And the objective is to find a permutation for jobs such that the total flowtime is minimized. Let $\pi=\{\pi_1, \pi_2, \ldots, \pi_n\}$ be a permutation of jobs and let $C(\pi_i, j)$ be defined as follows (Eq. (1)):

$$C(\pi_i, j) = \max\{C(\pi_{i-1}, j),\ C(\pi_i, j-1)\} + T_{\pi_i, j};\quad i = 1, 2, \ldots, n;\quad j = 1, 2, \ldots, m$$
$$\text{where } C(\pi_0, j) = 0 \text{ and } C(\pi_i, 0) = 0 \text{ for relevant } i \text{ and } j \tag{1}$$

Then the flowtime (or completion time) of a job π_i is defined to be $C(\pi_i, m)$, i.e, the completion time of job π_i on the last machine M_m. And the total flowtime (or total completion time) is defined as follows (Eq. (2)), i.e., the sum of the completion times of all jobs.

The permutation flowshop scheduling problem with total flowtime criterion was proved NP-hard by Garey et al. (1976)

$$C_{sum} = \sum_{i=1}^{n} C(\pi_i, m) \tag{2}$$

Therefore, instead of trying to find the optimal solution, many efforts had been devoted to design heuristics or metaheuristics, in order to find high-quality solutions within a reasonable computation time.

The heuristic methods can be classified into three categories: constructive heuristics (e.g. Wang et al., 1997), improvement heuristics (e.g. Ho, 1995), and composite heuristics that combine different heuristics. Rajendran and Ziegler (1997) presented a constructive heuristic incorporated with an improvement phase, which consists of one round of the insertion search. Woo and Yim (1998) also proposed a constructive heuristic, but they tried to find the best position to insert a selected job into the partial sequence. Liu and Reeves (2001) proposed a constructive heuristic and some composite heuristics that combined the proposed constructive heuristic with local search methods. They reported that the composite heuristics had better performance. Allahverdi and Aldowaisan (2002) applied the pair-wise exchange to existing heuristics to improve the performance. Framinan et al. (2005)

* Corresponding author at: Institute of Networking and Multimedia, National Chung Hsing University, 250 Kuo Kuang Road, Taichung, Taiwan.
Tel.: +886 4 22874020; fax: +886 4 22853869.
E-mail address: lytseng@cs.nchu.edu.tw (L.-Y. Tseng).

0925-5273/$ - see front matter © 2010 Elsevier B.V. All rights reserved.
doi:10.1016/j.ijpe.2010.05.003

gave a review of the heuristic methods for the total flowtime flowshop scheduling problem. In addition to the review, they also suggested two composite heuristics that outperformed other heuristics. From the trend of design of heuristics for this problem, it is noted that composite heuristics had better performance than constructive heuristics and improvement heuristics. Besides heuristics, recently, several metaheuristics were proposed for this problem.

In general, metaheurisitcs obtained better quality solutions than heuristics did, but metaheuristics needed more computation time. Yamada and Reeves (1998) presented a genetic local search algorithm for the total flowtime flowshop problem. Their algorithm spent much computation time and obtained good quality solutions. Rajendran and Ziegler (2004) proposed two ant colony optimization methods called M-MMAS and PACO for this problem. They compared the performance of their methods with that of Liu and Reeves (2001) and reported that for 83 out of the 90 benchmark problems taken from Taillard (1990), better solutions had been found by their methods. The next year, they proposed still other two ant colony optimization algorithms (Rajendran and Ziegler, 2005) that slightly improved the performance of M-MMAS and PACO. Tasgetiren et al. (2007) suggested a particle swarm optimization method. With this method, 57 out of 90 best known solutions reported by Liu and Reeves (2001), and Rajendran and Ziegler (2004) were improved. In this paper, we proposed a genetic local search algorithm with the orthogonal-array-based crossover operator and the orthogonal-array-based mutation operator for the total flowtime flowshop scheduling problem. With our method, 54 out of the 90 best known solutions reported by Liu and Reeves (2001), Rajendran and Ziegler (2004) and Tasgetiren et al. (2007) were improved. The other 23 out of the 90 achieved the same best known solutions. Also, with less computation time comparable to that used by Yamada and Reeves (1998), our method obtained better solutions than their method did.

The remaining parts of the paper are organized as follows. In Section 2, the concept of the orthogonal array is described. In Section 3, our genetic local search algorithm is presented. In Sections 4 and 5, experimental results and conclusions are given.

2. The orthogonal arrays

In this section, we briefly introduce the concept of orthogonal arrays which are used in experimental design methods. For more details, the reader may refer to (Montgomery (1991)). Suppose in an experiment, there are k factors and each factor has q levels. In order to find the best setting of each factor's level, q^k experiments must be done. Very often, it is not possible or cost effective to test all q^k combinations. It is desirable to sample a small but representative sample of combinations for testing. The orthogonal arrays were developed for this purpose. In an experiment that has k factors and each factor has q levels, an orthogonal array $OA(n, k, q, t)$ is an array with n rows and k columns, which is a representative sample of n testing experiments that satisfies the following three conditions. (1) For the factor in any column, every level occurs the same number of times. (2) For the t factors in any t columns, every combination of q levels occur the same number of times. (3) The selected combinations are uniformly distributed over the whole space of all the possible combinations. In the notation $OA(n, k, q, t)$, n is the number of experiments, k is the number of factors, q is the number of levels of each factor, and t is called the strength. Another often used notation for the orthogonal array is $L_n(q^k)$. In this notation t is omitted and is always set to 2. An $L_8(2^7)$ orthogonal array is shown in Table 1 as an illustrating example.

Table 1
$L_8(2^7)$ orthogonal array.

Test no.	Factors							Evaluation value (E_i)
	1	2	3	4	5	6	7	
1	0	0	0	0	0	0	0	E_1
2	0	0	0	1	1	1	1	E_2
3	0	1	1	0	0	1	1	E_3
4	0	1	1	1	1	0	0	E_4
5	1	0	1	0	1	0	1	E_5
6	1	0	1	1	0	1	0	E_6
7	1	1	0	0	1	1	0	E_7
8	1	1	0	1	0	0	1	E_8

For an experiment, there are various orthogonal arrays available. After an orthogonal array is chosen, one may apply the following criterion (the Taguchi method) to determine the best combinations of each factor's level in this experiment. Let E_i be the evaluation value of the ith experiment in the array. The main effect of factor j with level k, F_{jk} is defined by Eq. (3), where A_{ijk} is 1 if factor j's level is k in the ith experiment and A_{ijk} is 0 otherwise. After all F_{jk} had been computed, the level of factor j is chosen to be l if

$$F_{jl} = \max_{1 \le k \le q} F_{jk}.$$

$$F_{jk} = \sum_{i=1}^{n} E_i A_{ijk} \tag{3}$$

3. The proposed genetic local search algorithm

In this section, we described the proposed genetic local search (GLS) algorithm for the total flowtime flowshop scheduling problem. Our algorithm hybridizes the genetic algorithm and the tabu search. The genetic algorithm acts as a global search method in our algorithm, because the genetic algorithm is good at searching the whole solution space globally. The tabu search acts as the local search method, because the tabu search is good at searching the neighborhood of a solution. An orthogonal-array-based crossover operator (OA-crossover) and an orthogonal-array-based mutation (OA-mutation) were utilized in our algorithm to improve the performance. First, the proposed GLS algorithm is described in the following. The details of the OA-crossover, the OA-mutation, and the tabu search will be described in the subsections thereinafter.

/* Initialization */
Step 1: set the values of population size (P_s), mutation rate (P_m), termination condition (*Max_Stuck*), threshold for local search (TLS), and set *Stuck_Loop* to *0*.
Step 2: produce the initial population that consists of P_s randomly generated chromosomes.
Step 3: evaluate the total flowtime of each chromosome in the population. Deposit the chromosome with the best total flowtime in *BEST* and its total flowtime in *C**.
/* Crossover and selection */
Step 4: repeat Steps 5–7 P_s times.
Step 5: randomly choose two chromosomes P_1 and P_2. Apply OA-crossover to the parent chromosomes P_1 and P_2 to produce the child chromosome *Child*.
Step 6: if the total flowtime of *Child* is less than that of P_1 or P_2 and *Stuck_Loop* > *TLS*, then apply Local Search to *Child*.
Step 7: P_1 and P_2 are replaced by the best two of P_1, P_2, and *Child*.

Step 8: find the chromosome B with the best total flowtime C in the population. If $C < C^*$ then $C^* \leftarrow C$, $BEST \leftarrow B$ and $Stuck\_Loop \leftarrow 0$ else $Stuck\_Loop \leftarrow Stuck\_Loop+1$.
/* Mutation */
Step 9: randomly choose $P_s \times P_m$ chromosomes from the population and apply the OA-mutation to these chromosomes.
Step 10: if $Stuck\_Loop > Max\_Stuck$ then stop else go to Step 4.

The loop (Steps 5–7) performs the OA-crossover P_s times. A generation is viewed as including P_s times of the OA-crossover, the selection after the crossover, and the OA-mutation. In our experiments, threshold for local search (TLS) was set to 5. The local search is applied to the child chromosome after the OA-crossover, only if the total flowtime of the child chromosome is improved over P_1 or P_2, and the number of stuck generations (*Stuck_Loop*) is greater than *TLS* (five in our experiments). Because the local search is time consuming, we decided not to apply it when the OA-crossover alone is making progress. Only when the OA-crossover does not make progress for *TLS* generations, the local search is activated to help search. Also, if the child chromosome is not better than either of the parents, we do not waste time to apply the local search to such a not-so-promising child. The selection is a eugenic one. The child will replace parents only if it is better than one of the parents. If the number of stuck generations is more than *Max_Stuck* (10 in our experiments), the algorithm terminates.

3.1. Representation of chromosome and definition of fitness function

In genetic algorithms, a chromosome represents a solution in the solution space. For the permutation flowshop scheduling problem, we use a permutation π of jobs as a chromosome. For example, suppose there are six jobs and four machines in a flowshop scheduling problem. A permutation $\pi=[2, 3, 1, 6, 5, 4]$ is a permutation of six jobs and this chromosome represents a scheduling, in which the sequence of jobs on each machine is J_2, J_3, J_1, J_6, J_5, J_4.

The definition of fitness function is just the reciprocal of the objective function value, i.e., the reciprocal of the total flowtime of the scheduling represented by the chromosome.

3.2. OA–crossover

The crossover operator is used in genetic algorithms to find better solutions by recombining good genes from different parent chromosomes. One cut point or two cut points are usually used in the crossover operator. In contrast to this, multiple cut points are used in our crossover. Father chromosome and mother chromosome are randomly divided into multiple subsequences. Several recombinations of subsequences based on the orthogonal array are sampled and the Taguchi method is used to select the better subsequences for recombination. The OA-crossover had been used in Tsai et al. (2004) to solve the global numerical optimization problem and used in Ho and Chen (2000) to solve the traveling salesman problem. In this study, we use six cut points for twenty-job instances, fourteen cut points for fifty-job instances and thirty cut points for a-hundred-job instances. According to experiences, it is a good choice to increase the number of cut points in the OA-crossover as the size of the instance increases. The OA-crossover is described in the following.

Step 1: Let N be the number of pieces into which the user wants to cut parent chromosomes P_1 and P_2 for recombination. Generate the orthogonal array $L_{N+1}(2^N)$.

Step 2: Randomly choose $N-1$ cut points to cut P_1 and P_2 into N subsequences.

Step 3: Consult the ith row of the OA $L_{N+1}(2^N)$ and generate a sampled child C_i, for $i=1, 2, \ldots, N+1$. The jth subsequence of C_i is taken from the jth subsequence of P_1 if the level of the jth factor in row i of the OA is 0. Otherwise, the jth subsequence of C_i is taken from the jth subsequence of P_2. Repair C_i whenever it is necessary. (Repair scheme will be described later.)

Step 4: Calculate the evaluation value E_i of each sampled child C_i, for $i=1, 2, \ldots, N+1$. The evaluation value E_i is the fitness value (i.e. the reciprocal of the total flowtime) of the chromosome C_i.

Step 5: Calculate the main effect F_{jk} of factor j with level k, for $j=1, 2, \ldots, N$ and $k=0, 1$.

Step 6: Find the best level for each factor. The best level of factor j is k if $F_{jk}=max\{F_{j0}, F_{j1}\}$. Use the best levels of all factors to generate another child C_{N+2} (the Taguchi method). Repair C_{N+2} whenever it is necessary. Calculate the evaluation value E_{N+2}.

Step 7: From $C_1, C_2, \ldots, C_{N+2}$, choose the one with the best evaluation value to be the child chromosome.

The sample child C_i generated in Step 3 and the child C_{N+2} generated by the Taguchi method may need to be repaired. The following example illustrates the repair scheme. Fig. 1 shows two parent chromosomes Parent 1 and Parent 2 together with an OA $L_4(2^3)$. When generating the sampled child C_2, the second row of the OA $L_4(2^3)$ is consulted. Since the content of this row is 011, the first subsequence of C_2 is taken to be the first subsequence of Parent 1, which is 3, 5. The second subsequence of C_2 is taken to be the second subsequence of Parent 2, which is 7, 1. Up to now, it is all right and nothing to be repaired. The third subsequence of C_2 is taken to be the third subsequence of Parent 2, which is 4, 0, 5, 3. Now it needs to be repaired because 5, 3 already appeared in C_2. So the last two places of C_2 are cleared and the jobs not yet appearing in C_2 are taken from Parent 1 to put into these two places in the order they appear in Parent 1.

3.3. The local search scheme—tabu search

The tabu search was proposed by Glover (1986) in 1986. It uses the tabu list to remember those solutions that had been recently examined in order to avoid repeatedly searching the same region. The neighborhood structures are used in the tabu search. A neighborhood $N(x)$ of a solution x is defined as the set of all solutions that can be reached a form x by applying a transition operator once. The most common used transition operators are the exchange operator and the insertion operator. The exchange operator picks out two jobs from the job sequence and exchanges their locations. The insertion operator picks out one job from the sequence and inserts it into another location. We had compared the performance of these two operators and found that the insertion operator outperformed the exchange operator. Hence, the neighborhood structure used in the tabu search is based on the insertion operator. The tabu search used in our algorithm is described as follows. The tabu search begins the search with the given solution π with its total flowtime stored in C. We used π^* and C^* to store the current best solution and its total flowtime. NSP(α) which is the neighborhood search procedure will be described in Fig. 2.

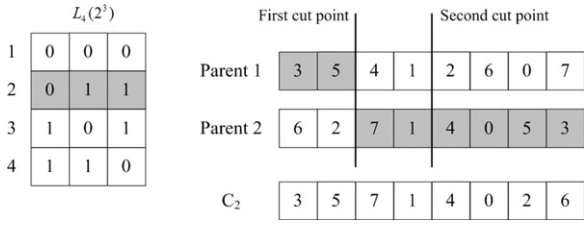

Fig. 1. An example that illustrates the repair scheme.

```
1. Procedure: Tabu Search(π, c)
2. begin
3.    π*←π;
4.    c*←c;
5.    generation ←0;
6.    stuck←0;
7.    While (generation<maxGeneration) do
8.       BestMove ← NSP(α);
9.       addToTabuList(BestMove);
10.      π← generateNewSchedule(BestMove);
11.      c← evaluation(π);
12.      if (c<c*) then
13.         π*←π;
14.         c*←c;
15.         stuck←0;
16.      else
17.         stuck←stuck+1;
18.         if (stuck >= maxStuck) then
19.            perturbation( );
20.            stuck←0;
21.         end if;
22.      end if;
23.      generation← generation+1;
24.   end while;
25.   return π*;
26. end;
```

Fig. 2. . The tabu search procedure.

We now describe the neighborhood search procedure NSP(α). α is a parameter defining the extent of the neighborhood, whose value is specified by the user. For example, if α is 3, each job that is picked out by the insertion operator can only be inserted into one of the three locations before or after its original location. In the neighborhood, NSP(α) will find the best move that is not forbidden by the tabu list. The length of the tabu list is seven and the scheme constructing the tabu list is that proposed by Nowicki and Smutnicki (1996). The aspiration criterion is that a solution which is better than the current best solution will not be forbidden by the tabu list.

If the solution found by NSP(α) does not improve the current best solution, the stuck count will be increased by one. The perturbation procedure will be activated when the stuck count reaches the value of *maxStuck*. The perturbation procedure randomly chooses one of the nine best moves other than the best move chosen by NSP(α) and applies the move to the current solution without considering if the move is forbidden by the tabu list or not.

3.4. OA-mutation

In this study, we proposed a novel OA-mutation operator as shown in Fig. 3. Let M be the user-specified maximum number of insertion operators that will be considered in constructing the mutation operator. Based on M, an orthogonal array $L_{M+1}(2^M)$ will be constructed and M randomly generated insertion operators will be considered as factors.

Each factor has two levels, namely 0 and 1. A factor with the level 0 means that the corresponding insertion is not chosen in the sampled mutation operator. Otherwise, the corresponding insertion operator is chosen in the sampled mutation operator. Similar to the construction of the OA-crossover, $M+1$ sampled mutation operator and another candidate mutation operator obtained by the Taguchi method will be derived and the best of them is taken as the best mutation operator. Note that the best mutation operator consists of several insertion operators. The OATest procedure in line 9 of Fig. 3 does the above mentioned

```
1. Procedure: OA-Mutation(π, c)
2. begin
3.    π*←π;
4.    c*←c;
5.    isImprove←false;
6.    generation ←0;
7.    stuck←0;
8.    While (generation<maxGeneration) do
9.       BestMutation← OATest(π, M);
10.      π'← generateTempSchedule(BestMutation);
11.      c'← evaluation(π');
12.      if (c'< c) then
13.         π←π';
14.         c←c';
15.         stuck←0;
16.         if (c'<c*) then
17.            π*←π';
18.            c*←c';
19.            isImprove←true;
20.         end if;
21.      else
22.         stuck←stuck+1;
23.         if (stuck >= maxStuck) and (c'< replaceRate * c*) then
24.            π←π';
25.            c←c';
26.            stuck←0;
27.         end if;
28.      end if;
29.      generation← generation+1;
30.   end while;
31.   if ( isImprove ) then
32.      return π*;
33.   else
34.      return π;
36.   end if;
37. end;
```

Fig. 3. . The OA-mutation procedure.

process to obtain the best mutation operator. In addition to the application of the orthogonal array to choose the best mutation operator among several candidate mutation operators, we apply the best mutation operator several times to constitute a trajectory search rather than just apply the best mutation operator once. This adds the intensification capability to the mutation operator.

In line 23 of the OA-mutation procedure, *replaceRate* is set to be 1.05 in our experiments. Under the condition that the number of stuck is greater than or equal to the value of *maxStuck*, the solution π' obtained by applying the best mutation operator to the current solution π will replace the current solution π if the total flowtime c' of π' is not worse than $1.05 \times c^*$, where c^* is the total flowtime of the current best solution π^*. Note that in this case c' is worse than the total flowtime c of the current solution π.

4. Experimental results

The proposed genetic local search algorithm had been implemented using C++ language on a personal computer, whose CPU is AMD K7 1.83 GHz and memory size is 512 MB and the operating system is Windows XP.

In the first experiment, the performance of the proposed GLS is compared with those of four other methods. These methods include the best heuristic method given in Liu and Reeves (2001), two ant colony algorithms, M-MMAS and PACO, proposed by Rajendran and Ziegler (2004), and the particle swarm optimization (PSO) suggested

Table 2
Parameter settings for the first experiment.

Number of jobs:	20	50	100
GLS			
Ps	50	200	200
Pm	0.1	0.1	0.1
TLS	5	5	5
Max-Stuck	10	10	10
N	7	15	31
Tabu search			
maxGeneration	50	50	50
maxStuck	5	5	5
α	3	5	5
OA-mutation			
maxGeneration	50	100	100
maxStuck	5	5	5
replaceRate	1.05	1.05	1.05
M	3	3	3

by Tasgetiren et al. (2007). The performance tests were conducted on 90 instances from the Taillard's benchmark (Taillard, 1990). Note that the best heuristic method given in Liu and Reeves (2001) is not always the same one for all 90 instances. In the above mentioned four methods, only Tasgetiren et al. (2007) (PSO) gave the CPU time they spent. Since they used a personal computer (with Intel P4 2.6 GHz CPU and 256 MB memory), whose computing power is comparable to that of ours, we set parameter values in such a way that our computing time is less than theirs. The parameter settings for the first experiment are given in Table 2.

The performance comparison is shown in Table 3 with the best ones printed in bold face. From Table 3, it is noted that 54 out of 90 best known solutions reported by Liu and Reeves (2001), Rajendran and Ziegler (2004), and Tasgetiren et al. (2007) were improved. On other 23 instances, our method obtained the same best known solutions.

Table 4 gives the CPU time comparison of the PSO (Tasgetiren et al., 2007) and our method. In the PSO, the CPU time is averaged

Table 3
Performance comparison with four other methods.

$n \times m$	LR Min	M-MMAS Min	PACO Min	PSO Min	GLS Min	Average
20 × 5	14,226	14,056	14,056	**14,033**	**14,033**	14,051
	15,446	**15,151**	15,214	**15,151**	**15,151**	15,216
	13,676	13,416	13,403	**13,301**	**13,301**	13,355
	15,750	15,486	15,505	**15,447**	**15,447**	15,476
	13,633	**13,529**	**13,529**	**13,529**	**13,529**	13,534
	13,265	13,139	**13,123**	**13,123**	**13,123**	13,135
	13,774	13,559	13,674	**13,548**	**13,548**	13,581
	13,968	13,968	14,042	**13,948**	**13,948**	13,954
	14,456	14,317	14,383	**14,295**	**14,295**	14,351
	13,036	12,968	13,021	**12,943**	**12,943**	12,969
20 × 10	21,207	20,980	20,958	**20,911**	**20,911**	20,945
	22,927	**22,440**	22,591	**22,440**	**22,440**	22,540
	20,072	**19,833**	19,968	**19,833**	**19,833**	19,854
	18,857	18,724	18,769	**18,710**	**18,710**	18,792
	18,939	18,644	18,749	**18,641**	**18,641**	18,705
	19,608	**19,245**	**19,245**	19,249	**19,245**	19,336
	18,723	18,376	18,377	**18,363**	**18,363**	18,403
	20,504	**20,241**	20,377	**20,241**	**20,241**	20,294
	20,561	**20,330**	**20,330**	**20,330**	**20,330**	20,352
	21,506	**21,320**	21,323	**21,320**	**21,320**	21,362
20 × 20	34,119	**33,623**	**33,623**	34,975	**33,623**	33,801
	31,918	31,604	31,597	32,659	**31,587**	31,601
	34,552	**33,920**	34,130	34,594	**33,920**	34,014
	32,159	31,698	31,753	32,716	**31,661**	31,727
	34,990	34,593	34,642	35,455	**34,557**	34,615
	32,734	32,637	32,594	33,530	**32,564**	32,590
	33,449	33,038	**32,922**	33,733	**32,922**	33,042
	32,611	32,444	32,533	33,008	**32,412**	32,480
	34,084	33,625	33,623	34,446	**33,600**	33,623
	32,537	32,317	32,317	33,281	**32,262**	32,309
50 × 5	65,663	65,768	65,546	65,058	**64,892**	65,076
	68,664	68,828	68,485	68,298	**68,132**	68,415
	64,378	64,166	64,149	63,577	**63,425**	63,863
	69,795	69,113	69,359	68,571	**68,478**	68,796
	70,841	70,331	70,154	69,698	**69,628**	69,758
	68,084	67,563	67,664	67,138	**67,109**	67,431
	67,186	67,014	66,600	66,338	**66,334**	66,664
	65,582	64,863	65,123	64,638	**64,459**	64,806
	63,968	63,735	63,483	63,227	**63,154**	63,288
	70,273	70,256	69,831	69,195	**69,184**	69,356
50 × 10	88,770	89,599	88,942	88,031	**87,944**	88,575
	85,600	83,612	84,549	83,624	**83,542**	84,040
	82,456	81,655	81,338	80,609	**80,558**	80,893
	89,356	87,924	88,014	87,053	**86,956**	87,511
	88,482	88,826	87,801	87,263	**87,002**	87,497
	89,602	88,394	88,269	87,255	**87,058**	87,631
	91,422	90,686	89,984	89,259	**89,196**	89,980

Table 3 (*continued*)

$n \times m$	LR Min	M-MMAS Min	PACO Min	PSO Min	GLS Min	Average
	89,549	88,595	88,281	**87,192**	87,358	87,936
	88,230	86,975	86,995	86,102	**85,975**	86,580
	90,787	89,470	89,238	88,631	**88,574**	89,450
50×20	129,095	127,348	**126,962**	128,622	127,011	127,700
	122,094	121,208	121,098	122,173	**120,104**	120,856
	121,379	118,051	117,524	118,719	**117,522**	117,928
	124,083	123,061	122,807	123,028	**120,983**	122,009
	122,158	119,920	**119,221**	121,202	119,319	119,852
	124,061	122,369	122,262	123,217	**121,393**	122,269
	126,363	123,609	125,351	125,586	**124,418**	125,128
	126,317	124,543	124,374	125,714	**123,937**	124,576
	125,318	124,059	123,646	124,932	**122,727**	123,424
	127,823	126,582	125,767	126,311	**125,126**	125,511
100×5	256,789	257,025	257,886	**254,762**	255,088	256,555
	245,609	246,612	246,326	245,315	**244,174**	245,174
	241,013	240,537	241,271	239,777	**239,641**	240,372
	231,365	230,480	230,376	**228,872**	228,941	229,706
	244,016	243,013	243,457	242,245	**241,726**	242,601
	235,793	236,225	236,409	234,082	**234,038**	234,888
	243,741	243,935	243,854	242,122	**241,611**	242,557
	235,171	234,813	234,579	232,755	**232,726**	233,487
	251,291	252,384	253,325	**249,959**	250,075	251,123
	247,491	246,261	246,750	**244,275**	244,888	246,120
100×10	306,375	305,004	305,376	303,142	**301,648**	303,414
	280,928	279,094	278,921	**277,109**	278,502	279,644
	296,927	297,177	294,239	292,465	**292,004**	293,269
	309,607	306,994	306,739	304,676	**304,215**	307,526
	291,731	290,493	289,676	288,242	**288,118**	289,376
	276,751	276,449	275,932	**272,790**	272,957	275,255
	288,199	286,545	284,846	**282,440**	282,685	284,325
	296,130	297,454	297,400	**293,572**	295,122	296,781
	312,175	309,664	307,043	305,605	**305,205**	307,181
	298,901	296,869	297,182	**295,173**	295,312	297,675
100×20	383,865	373,756	372,630	374,351	**372,405**	374,169
	383,976	383,614	381,124	**379,792**	380,290	382,053
	383,779	380,112	379,135	378,174	**376,796**	378,348
	384,854	380,201	380,765	380,899	**380,049**	381,695
	383,802	377,268	379,064	376,187	**375,873**	378,121
	387,962	381,510	380,464	379,248	**378,648**	380,811
	384,839	381,963	382,015	380,912	**378,691**	382,177
	397,264	393,617	393,075	392,315	**391,433**	393,051
	387,831	385,478	380,359	382,212	**381,638**	383,543
	394,861	387,948	388,060	386,013	**384,637**	386,841

Table 4
CPU time comparison between PSO and GLS in seconds and number of enumerated solutions comparison between GLS and M-MMAS and PACO.

$n \times m$	PSO CPU time	GLS CPU time	GLS Average number of enumerated solutions	M-MMAS and PACO Number of enumerated solutions
20×5	3.18	0.50	148,889.28	48,000
20×10	7.21	0.82	146,052.46	48,000
20×20	11.93	1.41	164,089.32	48,000
50×5	41.71	32.74	2,935,790.12	300,000
50×10	74.49	59.28	3,172,968.30	300,000
50×20	143.32	128.99	3,522,976.47	300,000
100×5	222.28	191.54	12,275,255.20	1,200,000
100×10	407.88	383.40	12,446,031.90	1,200,000
100×20	824.41	816.45	13,300,223.20	1,200,000

over ten instances with five runs for each instance and in the GLS, it is averaged over ten instances with ten runs for each instance. By inquiring Professor Rajendram, we learned that the number of solutions enumerated by M-MMAS or PACO is about 120^*n^*n, where n is the number of jobs. We also learned that the CPU time spent by M-MMAS or PACO for all 90 benchmark problems is less than 90 min using a personal computer with a Pentium-3 800 MHz CPU. Therefore, the average number of solutions enumerated by the GLS is also given in Table 4 as a comparison. From Table 4, the total CPU time needed to solve all 90 benchmark problems is 28.94 and 26.92 min for the PSO and the GLS, respectively. Although the number of enumerated solutions of M-MMAS or PACO is much less than that of the GLS, the CPU time spent by M-MMAS or PACO seems to be larger than that spent by the GLS, when taking into account the different clock rates of the CPUs used. This may be because an ant needs more time

in constructing a solution by choosing solution components step by step.

In the second experiment, the performance comparison is between the genetic local search algorithm proposed by Yamada and Reeves (1998) and our algorithm. Yamada and Reeves had tested their algorithm on 50 instances from the Taillard's benchmark (Taillard, 1990). These 50 instances include five sets, namely 20×5, 20×10, 20×20, 50×5, and 50×10, with 10 instances in each set. They used an HP workstation and had spent much more computation time to search more thoroughly. In the first three sets (20×5, 20×10, and 20×20), their algorithm obtained the same solutions as ours did, namely, those shown in Table 3. Their algorithm spent CPU time from several seconds to several minutes for each of these 30 instances. But our algorithm spent less than 1.5 s (as shown in Table 4) to obtain the same solutions. For the performance comparison of two algorithms on the fourth and the fifth sets, the parameter settings are listed in Table 5.

Yamada and Reeves conducted 10 runs on each of these 20 instances and the average CPU time spent is about 45 min for each instance in 50×5 and about 90 min for each instance in 50×10.

Table 5
Parameter settings for the second experiment.

$n\times m$	50×5	50×10
GLS		
Ps	1000	1000
Pm	0.1	0.2
TLS	5	5
Max-Stuck	20	20
N	15	15
Tabu search		
maxGeneration	100	100
maxStuck	5	5
α	5	10
OA-mutation		
maxGeneration	100	100
maxStuck	5	5
replaceRate	1.05	1.05
M	3	3

Table 6
Performance comparison of the YR's method and our method.

Instance		YR		GLS	
		Min	Average	Min	Average
ta031	50, 5	64,860	64,934.8	**64,838**	**64,934.5**
ta032		68,134	68,247.2	**68,114**	**68,208.4**
ta033		63,304	63,523.2	**63,258**	**63,387.1**
ta034		68,259	68,502.7	**68,226**	**68,361.9**
ta035		69,491	69,619.6	**69,397**	**69,516.6**
ta036		67,006	67,127.4	**66,865**	**67,007.8**
ta037		66,311	66,450	**66,261**	**66,335.5**
ta038		64,412	64,550.1	**64,371**	**64,538.6**
ta039		63,156	63,223.8	**62,981**	**63,096.4**
ta040		68,994	69,137.4	**68,906**	**69,019.8**
ta041	50, 10	87,430	**87,561.4**	**87,326**	87,627.2
ta042		83,157	**83,305.8**	**83,094**	83,434.8
ta043		79,996	80,303.4	**79,987**	**80,234.7**
ta044		86,725	**86,822.4**	**86,678**	86,907.3
ta045		**86,448**	**86,703.7**	86,530	86,841.4
ta046		86,651	**86,888**	**86,650**	86,917.3
ta047		**89,042**	**89,220.7**	89,178	89,434.4
ta048		86,924	87,180.5	**86,839**	**87,034.5**
ta049		85,674	**85,924.3**	**85,672**	86,018.7
ta050		88,215	**88,438.6**	**88,178**	88,658.1

We also conducted 10 runs on each of the 20 instances. But the average CPU time spent is 576.56 s (about 10 min) for each instance in 50×5 and 3865.69 s (about 64 min) for each instance in 50×10. The average number of solutions enumerated by GLS is 103,474,511.2 for each instance in 50×5 and 191,740,587.2 for each instance in 50×10. The comparison of solution quality is shown in Table 6. From Table 6, it is noted that 18 out of 20 best known solutions were improved.

5. Conclusions

In this study, a genetic local search algorithm for the flowshop scheduling problem with the total flowtime criterion was proposed. The proposed algorithm hybridized the genetic algorithm and the tabu search. It used the genetic algorithm to do the global search and the tabu search to do the local search. The OA-crossover operator was used to enhance the ability of intensification. Also, a novel OA-mutation operator was proposed. In addition to the ability of diversification in the traditional mutation operator, some capability of intensification was added in the proposed OA-mutation operator.

Experimental results showed that the proposed GLS algorithm improved 54 out of 90 current best solutions reported in the literature in short-term search and it also improved 18 out of 20 current best solutions reported in the literature in long-term search. Hence, the proposed GLS algorithm is very competitive in both the short-term and the long-term search. In the future, we plan to investigate the applicability of the proposed genetic local search algorithm to other scheduling problems. Specifically, two-machine flowshop scheduling problem had attracted more attention from researchers recently (Lee et al., 2010; Ladhari and Rakrouki, 2009). We also plan to investigate the applicability of the proposed algorithm to the two-machine flowshop scheduling problem.

Acknowledgements

The authors gratefully acknowledge the partial support by National Science Council of R.O.C under the contract NSC 98-2221-E005-049-MY3 and the partial support by the Ministry of Education, Taiwan, R.O.C. under the ATU plan.

References

Allahverdi, A., Aldowaisan, T., 2002. New heuristics to minimize total completion time in m-machine flowshops. International Journal of Production Economics 77, 71–83.

Framinan, J.M., Leisten, R., Ruiz-Usano, R., 2005. Comparison of heuristics for minimisation in permutation flowshops. Computers & Operations research 32, 1237–1254.

Garey, M.R., Johnson, D.S., Sethi, R., 1976. The complexity of flowshop and jobshop scheduling. Mathematics of Operations Research 1, 117–129.

Glover, F., 1986. Future paths for integer programming and links to artificial intelligence. Computers & Operations Research 13, 533–549.

Ho, J.C., 1995. Flowshop sequencing with mean flowtime objective. European Journal of Operational Research 81, 571–578.

Ho, S.Y., Chen, J.H., 2000. A GA-based systematic reasoning approach for solving traveling salesman problems using an orthogonal array crossover. in: Proceeding of the Fourth International IEEE Conference/Exhibition on High Performance Computing in Asia-Pacific Region, Beijing, China, pp. 659–663.

Johnson, S., 1954. Optimal two-and-three stage production schedules with setup times included. Naval Research Logistics Quarterly 1, 61–68.

Liu, J., Reeves, C.T., 2001. Constructive and composite heuristic solutions to the $P//\Sigma C_i$ scheduling problem. European Journal of Operational Research 132, 439–452.

Lee, W.C., Shiau, Y.R., Chen, S.K., Wu, C.C., 2010. A two-machine flowshop scheduling problem with deteriorating jobs and blocking. International Journal of Production Economics 124 (1), 188–197.

Ladhari, T., Rakrouki, M.A., 2009. Heuristics and lower bounds for minimizing the total completion time in a two-machine flowshop. International Journal of Production Economics 122 (2), 678–691.

Montgomery, D.C., 1991. Design and Analysis of Experiments third ed. Wiley, New York.

Nowicki, E., Smutnicki, C., 1996. A fast tabu search algorithm for the permutation flow-shop problem. European Journal of Operational Research 91, 160–175.

Rajendran, C., Ziegler, H., 1997. An efficient heuristic for scheduling in a flowshop to minimize total weighted flowtime of jobs. European Operational Research 103, 129–138.

Rajendran, C., Ziegler, H., 2004. Ant-colony algorithms for permutation flowshop scheduling to minimize makespan/total flowtime of jobs. European Journal of Operational Research 155 (2), 426–438.

Rajendran, C., Ziegler, H., 2005. Two ant-colony algorithms for minimizing total flowtime in permutation flowshops. Computer & Industrial Engineering 48, 789–797.

Taillard, E., 1990. Some efficient heuristic methods for the flowshop sequencing problem. European Journal of Operations Research 47, 65–74.

Tasgetiren, M.F., Liang, Y.C., Sevkli, M., Gencyilmaz, G., 2007. A particle swarm optimization algorithm for makespan and total flowtime minimization in the permutation flowshop sequencing problem. European Journal of Operational Research 177, 1930–1947.

Tsai, J.T., Liu, T.K., Chou, J.H., 2004. Hybrid Taguchi-genetic algorithm for global numerical optimization. IEEE Transactions on Evolutionary Computation 8 (4), 365–377.

Wang, C., Chu, C., Proth, J.-M., 1997. Heuristic approaches for $n/m/F//\Sigma C_i$ scheduling problems. European Journal of Operational Research 96, 636–644.

Woo, D.S., Yim, H.S., 1998. A heuristic algorithm for mean flowtime objective in flowshop scheduling. Computers & Operational Research 25, 175–182.

Yamada, T., Reeves, C.R., 1998. Solving the C_{sum} permutation flowshop scheduling problem by genetic local search. In: Proceedings of the 1998 IEEE International Conference on Evolutionary Computation, pp. 230–234.