European Journal of Operational Research 196 (2009) 869–876

Contents lists available at ScienceDirect

European Journal of Operational Research

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

Discrete Optimization

Hybrid genetic algorithm for permutation flowshop scheduling problems with total flowtime minimization

Yi Zhang *, Xiaoping Li, Qian Wang

School of Computer Science and Engineering, Southeast University, 210096 Nanjing, PR China
Key Laboratory of Computer Network and Information Integration, Southeast University, Ministry of Education, PR China

ARTICLE INFO

Article history:
Received 7 April 2007
Accepted 24 April 2008
Available online 4 May 2008

Keywords:
Genetic algorithm
Permutation flowshop
Total flowtime
Scheduling

ABSTRACT

In this paper, a HGA (hybrid genetic algorithm) is proposed for permutation flowshop scheduling problems (PFSP) with total flowtime minimization, which are known to be NP-hard. One of the chromosomes in the initial population is constructed by a suitable heuristic and the others are yielded randomly. An artificial chromosome is generated by a weighted simple mining gene structure, with which a new crossover operator is presented. Additionally, two effective heuristics are adopted as local search to improve all generated chromosomes in each generation. The HGA is compared with one of the most effective heuristics and a recent meta-heuristic on 120 benchmark instances. Experimental results show that the HGA outperforms the other two algorithms for all cases. Furthermore, HGA obtains 115 best solutions for the benchmark instances, 92 of which are newly discovered.

1. Introduction

The permutation flowshop scheduling problem (PFSP) is one of the most important combinational optimization problems, and can be described as follows: n different jobs are processed on m machines, with each job including m operations that are processed on different machines in the same order. Each operation has a predetermined processing time and each machine processes only one operation at a time, in the specified order. The goal is to find a sequence of jobs optimizing some criterion (or criteria). Minimization of makespan (Johnson, 1954; Nawaz et al., 1983; Ruiz et al., 2006), total tardiness (Potts and Van Wassenhove, 1985; Kim, 1995; Tan et al., 2000) and total flowtime (Rajendran and Ziegler, 1997; Framinan and Leisten, 2003; Li et al., in press) are general optimization objectives. The PFSP with multiple objectives has also been considered recently (Murata et al., 1996a; Ishibuchi and Murata, 1998; Ishibuchi et al., 2003). Multiple objective methodology is quite different from that for a single objective. Here we consider total flowtime minimization.

Total flowtime is an important performance measure which, when optimized, reflects a stable or uniform utilization of resources, a rapid turn-around of jobs, and minimization of in-process inventory (French, 1982; Rajendran, 1994). The PFSP with flowtime minimization is NP-hard, as shown by Gray et al. (1976), and heuristics and meta-heuristics are common methods. There are two kinds of heuristics, constructive and composite. The heuristics described by WY (Woo and Yim, 1998), RZ (Rajendran and Ziegler, 1997) and FL (Framinan and Leisten, 2003) are known to be efficient and constructive. Recently, many composite heuristics have been proposed, such as IH1–IH7 (Allahverdi and Aldowaisan, 2002), IH7-FL (Framinan and Leisten, 2003), FLR1 and FLR2 (Framinan et al., 2005), and ICH1–ICH3 (Li et al., in press). Meta-heuristics always find better solutions than heuristics. Therefore, meta-heuristics have been the focus of many researchers, with techniques such as the genetic algorithm (GA) (Reeves, 1995), Simulated Annealing (Osman and Potts, 1989), and Tabu Search (Tailard, 1990, 1993). This paper focuses on the GA.

For decades, many GAs for the PFSP have been proposed. Problems with makespan minimization are always studied. Reeves (1995) proposed a GA in which offspring only replace the chromosomes in the population with below average fitness, and "one-point crossover" was applied. Murata et al. (1996b) presented a GA with "two-point crossover", "shift mutation", and a local search. Recently, Ruiz et al. (2006) introduced a GA with a special crossover operator, in which identical jobs at the same positions in parent chromosomes are inherited by their offspring. Chang et al. (2007) proposed a GA for bi-criterion optimization called MGSPGA, in which MGS, presented by Chang et al. (2005) for TSP, makes it efficient.

In this paper, a hybrid GA (HGA) is proposed for PFSP with total flowtime minimization. LR(n/m), an effective seed sequence generator heuristic given by Liu and Reeves (2001), is adopted to produce one of the chromosomes in the initial population (the others are

* Corresponding author. Address: School of Computer Science and Engineering, Southeast University, 210096 Nanjing, PR China. Tel.: +86 02583790901.
E-mail addresses: zhangyiprivate@yahoo.com.cn (Y. Zhang), xpli@seu.edu.cn (X. Li), qwang@seu.edu.cn (Q. Wang).

0377-2217/$ - see front matter © 2008 Elsevier B.V. All rights reserved.
doi:10.1016/j.ejor.2008.04.033

generated randomly). An artificial chromosome is constructed by a weighted simple mining gene structure (WSMGS) based on the weighted mining gene structure (WMGS) given by Chang et al. (2007). A new crossover is presented by integrating the artificial chromosome with the Similar Job Order Crossover (SJOX) introduced by Ruiz et al. (2006). The new crossover inherits not only local good genes from the parents, but also global good genes from the artificial chromosome. Additionally, RZ (Rajendran and Ziegler, 1997) and FPE (Liu and Reeves, 2001) are applied as local search to improve generated chromosomes in every generation.

The rest of this paper is organized as follows: Section 2 gives a brief description of the PFSP with total flowtime minimization. HGA is introduced in Section 3, followed by parameter determination in Section 4. A full performance evaluation and comparison is shown in Section 5. Finally, conclusions are given in Section 6.

2. Problem description

The PFSP is a scheduling problem in which each job in $\Omega = \{J_1, \cdots, J_n\}$ is processed sequentially on m machines $M_1, \cdots, M_m$. Schedule π is a permutation of the n jobs, which can be denoted as $(\pi_{[1]}, \cdots, \pi_{[n]})$, in which $\pi_{[i]} \in \Omega$ is the ith $(i = 1, \cdots, n)$ job in π. Π is the set of all the permutations of the n jobs. To denote the start of any schedule, a dummy job $\pi_{[0]}$ is introduced with zero processing time. π can also be represented as $(\pi_{[0]}, \pi_{[1]}, \cdots, \pi_{[n]})$. Let $C_{i,\pi_{[k]}}$ denote the completion time of job $\pi_{[k]}$ on machine i. Assume $C_{i,\pi_{[0]}} = 0$ and let $t_{i,j}$ be the processing time of job j $(j = 0, 1, \cdots, n)$ processed on machine i $(i = 1, 2, \cdots, m)$. $C_{i,\pi_{[k]}}$ can be calculated as follows:

$$C_{i,\pi_{[k]}} = \begin{cases} C_{1,\pi_{[k-1]}} + t_{1,\pi_{[k]}}, & i = 1, \quad k = 1, \cdots, n \\ \max\{C_{i-1,\pi_{[k]}}, C_{i,\pi_{[k-1]}}\} + t_{i,\pi_{[k]}}, & i = 2, \cdots, m, \quad k = 1, \cdots, n. \end{cases} \tag{1}$$

Then, the total flowtime $(F(\pi))$ can be calculated by $F(\pi) = \sum_{k=1}^{n} C_{m,\pi_{[k]}}$. The goal is to find the optimal permutation π^* with total flowtime

$$F(\pi^*) = \min_{\pi \in \Pi}\{F(\pi)\} = \min_{\pi \in \Pi}\left\{\sum_{k=1}^{n} C_{m,\pi_{[k]}}\right\}. \tag{2}$$

3. Hybrid genetic algorithm

In this paper, a hybrid GA (HGA) is proposed for PFSP with total flowtime minimization. The mechanisms and operators of the HGA are given below.

3.1. Representation and selection mechanisms

Encoding the chromosome in a PFSP is relatively simple, which is the permutation order of jobs processed on machines. Chromosomes are randomly selected from the population for crossover and mutation.

3.2. Population initialization

In a traditional GA, each chromosome in the initial population is generated randomly. However, many experimental results have shown that the GA does not always result in good solutions due to the random method. The reason is that most of the randomly generated chromosomes have a poor total flowtime and thus pass unfavorable traits to their offspring. Reeves (1995) presented a population initialization method that leaded the success of the GA. In the method, one chromosome in the initial population is generated by NEH (Nawaz et al., 1983) and the others are randomly generated.

Liu and Reeves (2001) presented an effective method LR(x) to generate the initial solution for their composite heuristics, through which new best solutions for nearly all 120 benchmark instances (Tailard, 1993) were found. Additionally, according to Li et al. (in press), LR(n/m) is also a high performing initial solution generator for the iterative RZ method (Rajendran and Ziegler, 1997). Similar to the mechanism introduced by Reeves (1995), LR(n/m) can be used to generate one chromosome in the initial population of the HGA with the rest of the population randomly generated.

3.3. Crossover operator

Crossovers exchange the parent chromosome information in order to generate "better" offspring chromosomes, i.e. "better" solutions.

3.3.1. Simple mining gene structure

Mining gene structure (MGS) was proposed by Chang et al. (2005), who used statistic sorting of the elite chromosomes to find better genetic sources to manufacture the artificial chromosome for the traveling salesman problem (TSP) (Chang et al., 2007). MGS was adapted to the PFSP by Chang et al. (2007), and a "fabrication operator" was proposed to collect useful information from the elite chromosomes in the population, from which a set of artificial chromosomes are generated. The key to the operator is a "voting process" which can be described as follows: the number of each job in Ω at each position is counted and recorded in a $n \times n$ "dominance matrix" M. $M_{i,j}$ denotes the number of job i appearing at position j. With the "dominance matrix", artificial chromosomes can be easily generated by a "fabrication operator". Generally, more than one artificial chromosome is generated by the "fabrication operator", but only one is required in the HGA. In this paper, a simple mining gene structure (SMGS) is proposed which includes a "simple fabrication operator". Only one artificial chromosome is generated by the "simple fabrication operator". The formal description of SMGS is given as follows:

1. Voting
   For each elite chromosome π in the population
   For each position j in π
   if ($\pi_{[j]}$ is job i) $M_{i,j}++$;
2. $count \leftarrow 1$;
3. while ($count \leqslant n$)
   3.1 Find the element $M^*_{i,j}$ with the maximum value in the "dominance matrix" (just select the first one if there is more than one such element);
   3.2 Add job i at position j in the artificial chromosome;
   3.3 Set all elements in row i and column j as -1;
   3.4 $count++$;
4. Obtain the artificial chromosome.

After the artificial chromosome is generated, all values in the "dominance matrix" are reset to zero. The example for MGS in Chang et al. (2007) was performed to demonstrate the SMGS procedure as shown in Fig. 1. SMGS and MGS have the same "voting" process but different procedures to generate the artificial chromosome(s) from the "dominance matrix". Only one artificial chromosome is generated in SMGS whereas there are two in MGS.

3.3.2. Weighted simple mining gene structure (WSMGS)

Each elite chromosome in the current population votes to construct the "dominance matrix" to generate the artificial chromosome. However, it is unreasonable to assume every chromosome has the same contribution to the "dominance matrix" without con-

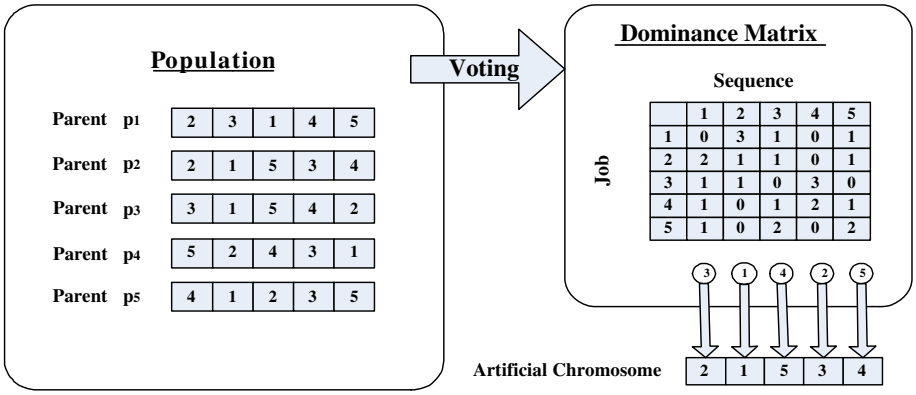

Fig. 1. Procedure of simple mining gene structure.

sideration of their fitness. Therefore, Chang et al. (2007) proposed a weighted mining gene structure (WMGS) in which chromosomes with higher fitness are given more weight in the "voting" procedure. In other words, the original vote of "1" is replaced by a new weighted one determined by the chromosome fitness. The calculation method proposed by Chang et al. (2007) is only suitable for PFSP with two optimization objectives. Consequently, a new weighted vote calculation method is presented in this paper for a single optimization objective, total flowtime minimization, as follows:

1. Generate an array of elite chromosomes in the current population by their non-decreasing fitness order.
2. Let P_{π^i} be the position of chromosome π^i in the array.
3. Calculate $M_{i,j} \leftarrow M_{i,j} + P_{\pi^i}$ for $i,j = 1, \cdots, n$.

The better the chromosome is, the bigger P_{π^i} is, and the larger the weight is. For the aforementioned example given in Fig. 1, assume that the total flowtime of the five chromosomes $\pi^1, \pi^2, \pi^3, \pi^4, \pi^5$ are 15, 34, 46, 38 and 24, respectively. After step 1, the order sequence is $(\pi^3, \pi^4, \pi^2, \pi^5, \pi^1)$, so P_{π^i} is (5, 3, 1, 2, 4) for $\pi^1, \pi^2, \pi^3, \pi^4, \pi^5$ and the artificial chromosome becomes (2, 1, 4, 3, 5) according to the weighted "dominance matrix" shown as follows:

$$\begin{bmatrix} 0 & 8 & 5 & 0 & 2 \\ 8 & 2 & 4 & 0 & 1 \\ 1 & 5 & 0 & 9 & 0 \\ 4 & 0 & 2 & 6 & 3 \\ 2 & 0 & 4 & 0 & 9 \end{bmatrix}$$

The weighted simple mining gene structure (WSMGS) can be established by replacing the weight of the SMGS with the result of the weighted vote calculation method.

3.3.3. Proposed crossover operator

During the evolution of the GA, chromosomes in the population have good genes resulting in high fitness. However, the genes are usually broken by traditional crossovers, such as "one-point crossovers" and "two-point crossovers". It is clear that performance can be improved by transferring these good genes to the next generation.

Definition 1. For a set of chromosomes, a job is called a Common Job if it locates at the identical slot (position) in all the chromosomes.

Ruiz et al. (2006) proposed four new crossover operators, SJOX (similar job order crossover), SBOX, SJ2OX and SB2OX, which are based on the traditional crossover operators, "one-point crossover" or "two-point crossover". SJOX is an extension of "one-point crossover", which can be described as follows:

1. Generate a random integer x $(1 \leqslant x \leqslant n)$ as a "crossover point"; Search all Common Jobs in the parent (Father and Mother) chromosomes (CJ(FM) for short);
2. Common Jobs are inherited by son and daughter chromosomes directly, positions unchanged.
3. The son inherits all the unassigned jobs before the "crossover point" from the father chromosome and the daughter from the mother.
4. The other elements of the son after the "crossover point" are copied in the same order from his mother chromosome and the daughter elements from her father.

SJOX transfers good genes from parent chromosomes to their offspring directly. To get better chromosomes, good global genes should also be mined for inheritance by the next generation. In the proposed crossover operator, the artificial chromosome produced by WSMGS is integrated with SJOX. In other words, in addition to the genes (or CJ(FM)) passed by a set of parents to their offspring in SJOX, Common Jobs between the Artificial and the Father (CJ(AF) for short) chromosomes are inherited by the son, and those between the Artificial and the Mother (CJ(AM) for short) chromosomes by the daughter. This procedure can be stated as follows:

1. Generate a random integer x $(1 \leqslant x \leqslant n)$ as "crossover point";
2. Search Common Jobs CJ(FM), CJ(AF) and CJ(AM);
3. Transfer Common Jobs CJ(FM) and CJ(AF) to the son chromosome with positions unchanged. Transfer Common Jobs CJ(FM) and CJ(AM) to the daughter chromosome with positions unchanged.
4. The son inherits all the unassigned jobs before the "crossover point" from the father chromosome and the daughter from the mother.
5. The other elements of the son after the "crossover point" are copied in the same order from his mother chromosome and the daughter's from her father.

This new crossover operator is similar to "SJOX" except for the integration of information from the artificial chromosome. Though artificial chromosomes generated by WMGS are also involved in the evolution of the GA described by Chang et al. (2007), they are just individuals in the next population. Effectiveness of the GA was improved in this way. However, artificial chromosomes exert an influence on populations in an indirect and implicit way. In

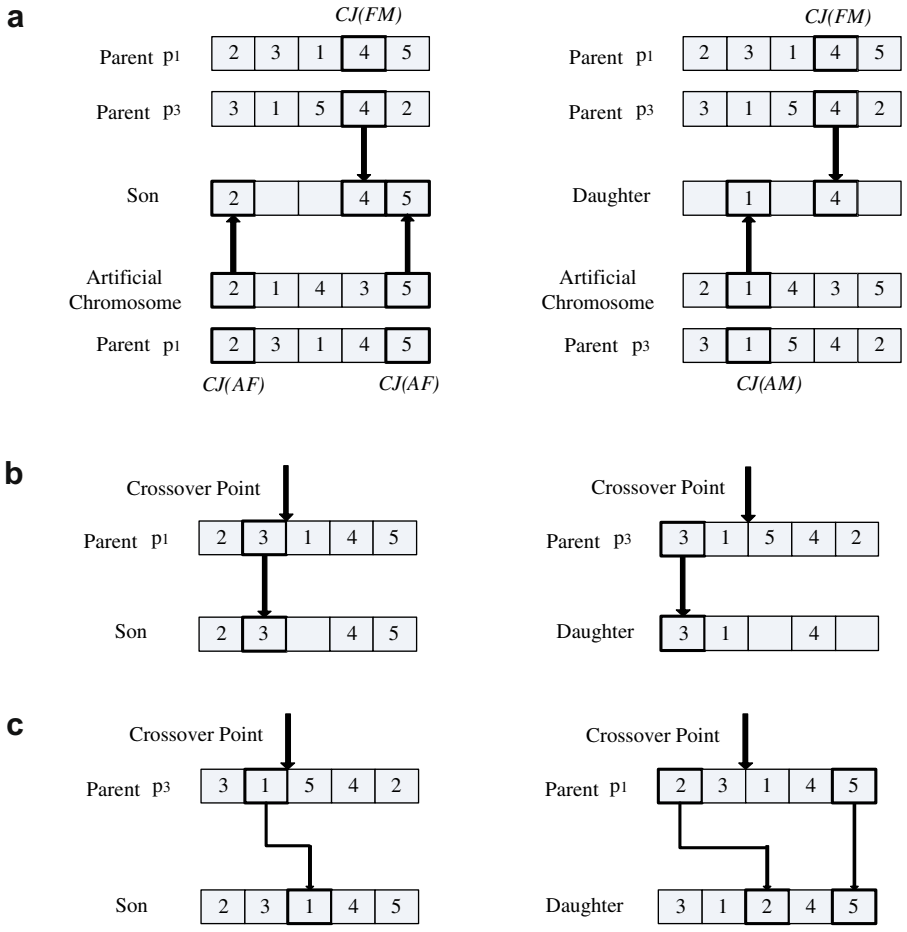

Fig. 2. Steps of the proposed crossover operator.

our proposed crossover operator, the artificial chromosome participates in crossover directly and mines more good genes (both local and global genes). Therefore, many more good genes can be inherited by offspring to make the HGA more effective.

For example, consider the artificial chromosome (2,1,4,3,5) constructed by WSMGS. π_1, π_5 are selected as parent chromosomes. Let $x = 2$ and see Fig. 2 for steps 3–5.

3.4. Mutation

After several generations, chromosomes in the population are similar enough to each other such that only local optimization may be possible. Mutation is one of the operators to avoid local optimization and only a slight gene change is needed for a chromosome. Murata et al. (1996b) and Nearchou (2004) evaluated different mutation operators for PFSP with makespan minimization. Both of them showed that shift mutation is the best and thus it is adopted in the proposed HGA.

3.5. Termination criteria

In traditional GA, either computation time or the number of generations is selected as termination criterion. In this paper, both of them are considered. If a solution cannot be improved any more in consecutive *COUNT* generations, the algorithm terminates. However, it is very time-consuming for large problems, such as 500×20. Therefore, limited computation time, $5 \times n \times m$ seconds for example, is chosen as the second termination criterion. In other words, HGA terminates when either of the two conditions is true.

3.6. Local search

Li et al. (in press) proposed three composite heuristics (ICH1–ICH3) based on three simple heuristics: LR(n/m) (Liu and Reeves, 2001), RZ (Rajendran and Ziegler, 1997) and FPE (Liu and Reeves, 2001). ICH1–ICH3 are very efficient and are effective for total flowtime minimization with some large problems. Twenty-six out of 30 best solutions were improved for the last 30 (from 91 to 120) Taillard (1993) benchmark instances. Experimental results show that RZ and FPE are very effective at locally improving a solution. Therefore, RZ and FPE are used to improve each generated chromosome π in every generation, i.e. π is improved by RZ and FPE sequentially.

3.7. HGA description

Notations are given before the proposed HGA description:

S	population size
P_c	probability of crossover
P_m	probability of mutation
COUNT	limitation of consecutive generations for solution not improved
count	record of consecutive generations for solution not improved
P_e	value is between 0 and 1, and the most best $P_e \times S$ in population are selected as elite chromosomes
E	set of elite chromosomes
π^*	current best solution
$\pi^\#$	artificial chromosome generated by WSMGS

HGA is described as follows:

```
{
  count ← 1;
  population ←PopulationInitialization(S);
  PopulationEvaluation(population);
  TFT_min ← MinTFT(population);
  π* ← MinTFTPermutation(population);
  E ← EliteSelection(P_e, population);
  π# ← WSMGS(E);
  while(count ⩽ COUNT AND within CPU time limitation)
  {
    Create offspringPopulation without any chromosome
    Repeat until covering the entire population
    {
      father←RandomSelect(population);
      mother←RandomSelect(population);
      son←Crossover(father,mother,π#,P_c);
      daughter ←Crossover(mother,father,P_c);
      Mutation(son,P_m);
      Mutation(daughter,P_m);
      //local search to improve both son and daughter
      RZ_FPE(son);
      RZ_FPE(daughter);
      Add(son,offspringPopulation);
      Add(daughter,offspringPopulation);
    }
    //population updated by generational schema in Section
    3.5
    population
    ←PopulationUpdate(population,offspringPopulation);
    PopulationEvaluation(population);
    E ←EliteSelection(P_e,population);
    π# ← WSMGS(E);
    if (MinTFT(population) < TFT_min)
    {
      TFT_min ← MinTFT(population);
      π* ← MinTFTPermutation(population);
      count ← 1;
    }
    else
      count + +;
  }
  return TFT_min;}
```

4. Parameter determination

In this section, two important parameters are determined. One is the number of elite chromosomes used in WSMGS and the other is the *COUNT* for termination criteria.

4.1. The number of elite chromosomes

P_e is very important for the quality of the artificial chromosome. "Bad" chromosomes in the population participate in the voting process and thus, the artificial chromosome yielded is not good if P_e is large. On the contrary, information for generating a reasonable artificial chromosome is not enough and HGA would be premature or represent local optimization. In this paper, P_e was determined by the following experiment:

Some parameters were initialized as $S = 50$, $P_c = 1.0$, $P_m = 0.02$, $COUNT = 10$ and $P_e \in \{0.2, 0.4, 0.5, 0.6, 0.8\}$. It is necessary for each chromosome to pass all its good genes to its offspring. In other words, each chromosome in the parent population should participate in crossover and transfer good genes to the offspring. Thus $P_c = 1.0$. Additionally, the bigger *COUNT* is, the higher the probability of a good solution. In this experiment, *COUNT* was initially set to the relatively large value of 10.

To obtain an appropriate P_e, 90 benchmark instances were tested. The following average relative percentage deviation (ARPD) defined by Ruiz et al. (2006) was adopted to evaluate performance:

$$\mathrm{ARPD} = \sum_{i=1}^{R} \left(\frac{(S_i - S_{\mathrm{best}}) \times 100}{S_{\mathrm{best}}} \right) / R \tag{3}$$

In Eq. (3), S_i denotes the solution generated by HGA, R is the number of replications for each instance and S_{best} represents the best result during the R tests. In the experiment, each instance was repeated 5 times, i.e. $R = 5$. Experimental results are shown in Table 1 with the smallest ARPD with different P_e for each problem labeled in bold.

Table 1
Performance of 90 benchmark instances with different P_e

Problem	$P_e = 0.2$	$P_e = 0.4$	$P_e = 0.5$	$P_e = 0.6$	$P_e = 0.8$
20×5	0.1265	0.1021	**0.0821**	0.1132	0.1483
20×10	0.1840	0.1506	0.1236	**0.1076**	0.1484
20×20	0.2698	0.2260	0.2163	**0.0949**	0.1929
50×5	0.3562	**0.2426**	0.3274	0.3108	0.2817
50×10	0.3758	0.3728	0.3684	0.4193	**0.3090**
50×20	0.3610	0.4055	0.3314	**0.2026**	0.3841
100×5	0.3421	0.3631	0.3086	0.2018	**0.1752**
100×10	0.3421	0.2874	0.2475	**0.1845**	0.4027
100×20	0.2578	0.3142	0.2483	**0.2014**	0.3014

Table 1 illustrates that $P_e = 0.6$ is the best for problems with a size of 20×10, 20×20, 50×20, 100×10 and 100×20, whereas it was the worst for a 50×10 problem. Nevertheless, $P_e = 0.6$ was still selected since it is the best for more than 50% (5 out of 9) of the total number of problems.

4.2. COUNT determination

COUNT is important to HGA performance. In fact, the solution can not be improved if it has not been improved for several consecutive generations, owing to its high fitness. If *COUNT* is large, little improvement is made during the last several iterations and the efficiency of HGA decreases. If it is small, the HGA terminates quickly without obtaining good solutions. Therefore, it is essential to determine a reasonable *COUNT*.

As described in Section 4.1, P_e is 0.6. The other parameters are initially set at $S = 50$, $P_c = 1.0$, $P_m = 0.02$ and $COUNT = 10$. To illustrate the degree of improvement of π^*(the current best solution) in every generation, $A[COUNT]$ is generated. $A[i]$ $(i = 1, \cdots, COUNT)$ records the number of π^* improved after i successive generations for an instance. For the evolution procedure depicted in Fig. 3, π^* is

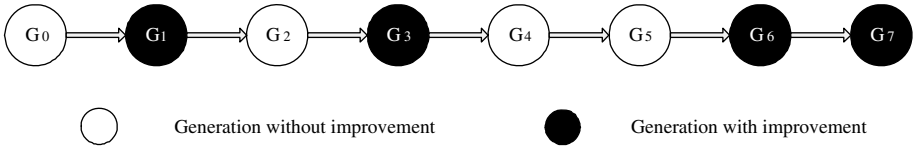

Fig. 3. Evolution procedure of the current best solution.

Table 2
Numbers of current best solution improved

Problem	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]	A[9]	A[10]
20×5	22	6	1	1	0	0	0	0	0	0
20×10	31	1	6	0	0	0	0	0	0	0
20×20	38	8	3	0	0	1	0	0	0	0
50×5	78	16	12	4	1	2	0	0	0	0
50×10	85	27	14	2	2	0	0	0	0	0
50×20	93	22	4	5	4	3	0	0	0	0
100×5	113	32	16	8	3	5	0	0	0	0
100×10	138	28	13	4	5	4	1	0	0	0
100×20	156	34	20	6	5	5	0	0	0	0
Total	754	183	89	30	20	20	1	0	0	0

Table 3
Performance comparisons without local search on benchmark instances

Problem	HGA_NLS	CGA	PSO_NLS
20×5	**1.151**	4.719	1.259
20×10	1.330	3.744	**0.771**
20×20	1.731	2.786	**0.950**
50×5	**0.124**	10.184	2.950
50×10	**0.324**	8.054	2.391
50×20	**0.393**	6.241	1.693
100×5	**0.006**	14.491	6.192
100×10	**0.160**	12.943	5.293
100×20	**0.081**	8.494	3.426
Average	**0.589**	7.962	2.769

improved a total of 4 times. $A[1] = 2$ because π^* is improved from G_0 to G_1 and from G_6 to G_7 (G_i represents the ith generation). $A[2] = 1$ for π^* improved from G_1 to G_3 via G_2, i.e., G_2 makes no improvement. π^* is improved from G_3 to G_6 via G_4 and G_5, in both of which no improvement is yielded and $A[3] = 1$.

For the above 90 benchmark instances, numbers of the current best solution being improved are shown in Table 2, which illustrates that $A[i] = 0$ $(i = 8, 9, 10)$ for all instances and $A[7] \neq 0$ for only one case (100×10). In other words, the current best solution can only be slightly improved after six successive generations. Therefore, a value of 6 can be assigned to *COUNT* because it is a waste of time to produce any more generations. Although little improvement can be made for 20×5, 20×10 and 20×20 cases, $COUNT = 6$ is also acceptable because it is fast for small problems.

5. Computational experiments

To evaluate the performance of the proposed HGA, the HGA without local search (HGA_NLS for short) is compared with one of the classical GAs with "Two-point crossover" and "shift mutation" (CGA for short) and PSO without local search (PSO_NLS for short). PSO for PFSP with total flowtime or makespan minimization was recently proposed by Tasgetiren et al. (2007), and is a stochastic optimization approach mimicking the process of a swarm of bees looking for the best location for their honeycomb. The PSO_NLS parameters are identical to those in Tasgetiren et al. (2007), i.e. population size is twice as the number of jobs, $c_1 = c_2 = 2$, $w^0 = 0.9$ and never decreased below 0.4, $\beta = 0.975$ and the termination criterion is 500 generations. For performance comparison, HGA_NLS and CGA have the same termination criterion, with parameters $S = 50$, $P_c = 1.0$, $P_m = 0.02$, $P_e = 0.6$, $COUNT = 6$ for HGA_NLS and $S = 50$, $P_c = 0.8$, $P_m = 0.02$, $COUNT = 6$ for CGA.

ARPD, defined in Eq. (3), is still used to evaluate performance. S_i is the solution generated by CGA, PSO_NLS or HGA_NLS and S_{best} represents the best solution yielded by the three algorithms. All algorithms were implemented in Java and tested on a 2.93GHZ P4 PC with 512M RAM. All algorithms were run five times. The first 90 Tailard benchmark instances were tested and the experimental results are given in Table 3, in which we can see that HGA_NLS is better than the other two algorithms, and CGA is the worst. The ARPD of HGA_NLS is stable whereas those of CGA and PSO_NLS are variable. PSO_NLS is suitable for small problems.

To evaluate the performance of the HGA, the HGA was compared with ICH2 and PSO_LS (PSO with local search), two current highly effective algorithms. ICH2 (Li et al., in press) seems to be one of the best composite heuristics so far.

ARPD is still used to evaluate performance. S_i is the solution generated by ICH2, PSO_LS or HGA. S_{best} represents the best solution yielded by the three algorithms. Let $\bar{t}$ denote the average time an algorithm is required. All 120 benchmark instances were run five times for both PSO_LS and HGA, and once for ICH2 because it is a deterministic heuristic. The best solutions over the five replications were recorded. Parameters for PSO_LS and HGA were identical to those in PSO_NLS and HGA_NLS, respectively. Experimental results are given in Table 4.

Table 4 shows that PSO_LS spent over 78,000 seconds on Ta111 (with a size of 500×20), which was very time-consuming, so the last nine instances were not performed (the corresponding ARPD and $\bar{t}$ are not provided in Table 4). In fact, the time required by the HGA with the same last ten largest instances would be too long as well if there were no limitations on computation time. Consequently, the last ten instances were tested by HGA just once. Nevertheless, the HGA obtains better solutions than ICH2 for the last group (instances 111–120), so its ARPD is zero.

Table 4 shows that the average performance of HGA was better than that of the other two algorithms. The ARPD of the HGA for each group was less than 0.4% (the smallest ARPD for each group is highlighted in bold). It should be noted that the ARPD for 200×10 and 200×20 were 0.182 and 0.179, respectively, which is very good for such complex and large problems. This is because the termination criterion of HGA is dynamic, i.e. HGA terminates if no improvement can be made in six successive generations (the computation time termination condition is rarely satisfied). However, many other GAs adopt static termination criteria such as a fixed amount of computation time or constant total number of generations, which should be limited to a relatively small value for complex and large problems. Otherwise, time may be wasted without solution improvement. In other words, HGA tries to find the best solution possible (it does not terminate if improvement can be made) and manages to terminate as soon as possible (it terminates if no improvement can be made in six successive generations). The HGA outperforms both ICH2 and PSO_LS for all the problems except the last group.

Table 4
Performance comparison on Tailard benchmark instances

Problem	ICH2		PSO_LS		HGA	
	ARPD (%)	$\bar{t}$ (seconds)	ARPD (%)	$\bar{t}$ (seconds)	ARPD (%)	$\bar{t}$ (seconds)
20×5	1.034	0.015	0.172	2.276	**0.025**	2.175
20×10	1.308	0.016	0.308	3.472	**0.055**	4.136
20×20	1.447	0.031	0.384	6.065	**0.033**	7.574
50×5	1.275	0.234	0.828	23.098	**0.144**	40.854
50×10	1.927	0.360	1.378	41.001	**0.222**	111.743
50×20	2.395	0.578	1.652	79.840	**0.288**	189.980
100×5	0.765	2.234	0.981	155.439	**0.125**	461.329
100×10	1.512	3.765	1.379	297.054	**0.230**	826.764
100×20	2.353	5.603	1.588	580.393	**0.336**	2014.957
200×10	0.855	41.406	1.416	2153.648	**0.182**	8515.007
200×20	1.337	72.125	1.287	4472.054	**0.179**	17849.583
500×20	0.160	1987.324	–	–	**0.0**	50000
Average	1.364		1.034		**0.152**	

In fact, most computation time is spent on the local search operation, which is applied to all offspring of HGA. To demonstrate the effectiveness of the proposed HGA with local search for all offspring, the HGA was compared with the other five hybrid GAs with local search for just partial offspring. In addition to HGA_NLS, mentioned above, the other four are defined as follows:

HGA_BI only better child chromosome is improved by local search
HGA_RI only one randomly selected child chromosome is improved by local search
HGA_2G local search is applied every two generations
HGA_3G local search is applied every three generations

HGA_BI and HGA_RI are different from HGA_2G and HGA_3G in local search pattern. The former two apply local search horizontally and the latter two vertically. The six algorithms are performed on the first 90 Tailard benchmark instances and the ARPD results shown in Table 5.

Table 5 implies that HGA obtains the best ARPD both on average and for each group while HGA_NLS had the worst. In other words, local search can improve performance. The ARPD of the HGA was about half of that of either HGA_BI or HGA_RI and one third of those of both HGA_2G and HGA_3G, i.e., it is better apply local search on all offspring than partial offspring. The HGA requires more computation time than HGA_BI and HGA_RI, however, HGA terminates with fewer generations so the overall computation time of HGA was slightly more than that of HGA_BI or HGA_RI. This is similar to HGA_2G and HGA_3G. Therefore, HGA is adopted in this paper.

Furthermore, to illustrate the influence of LR(n/m), local search, and global gene inheritance on HGA, HGA was compared with

Table 5
Performance comparison for different local search patterns

Problem	HGA	HGA_BI	HGA_RI	HGA_NLS	HGA_2G	HGA_3G
20 × 5	**0.025**	0.085	0.078	2.127	0.076	0.127
20 × 10	**0.051**	0.080	0.087	2.513	0.114	0.122
20 × 20	**0.033**	0.094	0.071	2.385	0.080	0.107
50 × 5	**0.176**	0.322	0.344	2.171	0.389	0.516
50 × 10	**0.293**	0.530	0.521	4.035	0.599	0.743
50 × 20	**0.346**	0.594	0.583	4.548	0.598	0.740
100 × 5	**0.132**	0.292	0.308	1.169	0.377	0.467
100 × 10	**0.297**	0.421	0.540	2.988	0.570	0.640
100 × 20	**0.355**	0.596	0.645	4.865	0.804	0.949
Average	**0.190**	0.335	0.353	2.967	0.401	0.490

Table 6
Comparison between HGA and those with different operator

Problem	HGA	HGA_NLS	HGA_NLR	HGA_NG
20 × 5	0.025	2.127	0.023	0.022
20 × 10	0.055	2.517	0.050	0.047
20 × 20	0.033	2.385	0.038	0.030
50 × 5	0.263	2.260	0.236	0.353
50 × 10	0.369	4.114	0.363	0.405
50 × 20	0.374	4.577	0.383	0.422
100 × 5	0.249	1.286	0.252	0.256
100 × 10	0.365	3.059	**0.432**	**0.393**
100 × 20	0.422	4.835	0.518	0.532
Average	0.239	3.018	0.255	0.273

Table 7
Performance comparisons in terms of the existing best solutions

Problem	ARPD (%)	Problem	ARPD (%)
20 × 5	0.025	100 × 5	−0.016
20 × 10	0.055	100 × 10	−0.117
20 × 20	−0.035	100 × 20	−0.748
50 × 5	0.053	200 × 10	−0.462
50 × 10	0.016	200 × 20	−0.806
50 × 20	−0.625	500 × 20	−0.129
Average		−0.232	

Table 8
Best solutions for Tailard benchmark instances

Problem no.	Best solution found	Heuristics getting the solution	Problem no.	Best solution found	Heuristics getting the solution
Ta001	14033	PSO, HGA	Ta061	254762	PSO
Ta002	15151	M-MMAS, HGA, PSO	Ta062	243850	HGA
Ta003	13301	PSO, HGA	Ta063	239173	HGA
Ta004	15447	PSO, HGA	Ta064	228705	HGA
Ta005	13529	M-MMAS, PACO, HGA, PSO	Ta065	241432	HGA
Ta006	13123	PACO, PSO, HGA	Ta066	233698	HGA
Ta007	13548	PSO, HGA	Ta067	241650	HGA
Ta008	13948	PSO, HGA	Ta068	232734	HGA
Ta009	14295	PSO, HGA	Ta069	249920	HGA
Ta010	12943	PSO, HGA	Ta070	244131	HGA
Ta011	20911	PSO, HGA	Ta071	300507	HGA
Ta012	22440	M-MMAS, HGA, PSO	Ta072	277109	PSO
Ta013	19833	M-MMAS, HGA, PSO	Ta073	290468	HGA
Ta014	18710	PSO	Ta074	303443	HGA
Ta015	18641	PSO, HGA	Ta075	286647	HGA
Ta016	19245	M-MMAS,PACO	Ta076	272764	HGA
Ta017	18363	PSO, HGA	Ta077	282373	HGA
Ta018	20241	M-MMAS, HGA, PSO	Ta078	293067	HGA
Ta019	20330	M-MMAS, PACO, HGA, PSO	Ta079	304330	HGA
Ta020	21320	M-MMAS, HGA, PSO	Ta080	293932	HGA
Ta021	33623	M-MMAS, PACO, HGA, PSO	Ta081	369652	HGA
Ta022	31587	HGA	Ta082	376067	HGA
Ta023	33920	M-MMAS, HGA, PSO	Ta083	373199	HGA
Ta024	31661	HGA	Ta084	374832	HGA
Ta025	34557	HGA	Ta085	371268	HGA
Ta026	32564	HGA	Ta086	375463	HGA
Ta027	32922	PACO, HGA	Ta087	376353	HGA
Ta028	32412	HGA	Ta088	387189	HGA
Ta029	33600	HGA	Ta089	378657	HGA
Ta030	32262	HGA	Ta090	382004	HGA
Ta031	64956	HGA	Ta091	1050564	HGA
Ta032	68298	PSO	Ta092	1040604	HGA
Ta033	63513	HGA	Ta093	1050785	HGA
Ta034	68571	PSO	Ta094	1038885	HGA
Ta035	69562	HGA	Ta095	1041510	HGA
Ta036	67111	HGA	Ta096	1013421	HGA
Ta037	66318	HGA	Ta097	1061285	HGA
Ta038	64418	HGA	Ta098	1049007	HGA
Ta039	63157	HGA	Ta099	1026991	HGA
Ta040	69141	HGA	Ta100	1038016	HGA
Ta041	87593	HGA	Ta101	1235238	HGA
Ta042	83517	HGA	Ta102	1254529	HGA
Ta043	80316	HGA	Ta103	1274798	HGA
Ta044	86953	HGA	Ta104	1245656	HGA
Ta045	86905	HGA	Ta105	1236246	HGA
Ta046	87008	HGA	Ta106	1237754	HGA
Ta047	89155	HGA	Ta107	1248821	HGA
Ta048	87192	PSO	Ta108	1249644	HGA
Ta049	86086	HGA	Ta109	1237428	HGA
Ta050	88363	HGA	Ta110	1253075	HGA
Ta051	125850	HGA	Ta111	6723143	HGA
Ta052	119442	HGA	Ta112	6844840	HGA
Ta053	117315	HGA	Ta113	6772110	HGA
Ta054	121114	HGA	Ta114	6809460	HGA
Ta055	118975	HGA	Ta115	6742209	HGA
Ta056	120955	HGA	Ta116	6729388	HGA
Ta057	123740	HGA	Ta117	6706950	HGA
Ta058	122940	HGA	Ta118	6795769	HGA
Ta059	122123	HGA	Ta119	6736573	HGA
Ta060	124936	HGA	Ta120	6764295	HGA

Note: M-MMAS and PACO are ant-colony algorithms given by Rajendran and Ziegler (2004).

HGA_NLR (HGA without LR(n/m)),HGA_NLS and HGA_NG (HGA without global genes inheritance, i.e. SJOX). Table 6 shows the ARPD results on the first 90 Tailard benchmark instances.

Table 6 illustrates that on average, the ARPD of HGA is the best whereas that of HGA_NLS is the worst. In other words, local search exerts the most favorable influence on HGA performance. The HGA is better than both HGA_NLR and HGA_NG, i.e., LR(n/m) and global gene transference are beneficial to population initialization and crossover, respectively. The ARPD of both HGA and HGA_NLR was more stable than that of HGA_NG except for 100×10 problems (highlighted in Table 6), i.e., global gene inheritance can maintain the solution stability.

To further evaluate performance of HGA, solutions yielded by HGA were compared with the best known solutions. Li et al. (in press) listed the existing best total flowtimes for all 120 benchmark instances and Tasgetiren et al. (2007) independently gave the best ones for the first 90 instances. ARPD is still used in which S_i denotes the solution generated by HGA, but S_{best} represents the best result of the two algorithms. Results are shown in Table 7.

Table 7 illustrates that the performance of the HGA improves as the overall problem size increases. For example, the ARPD of the proposal is only 0.025% for the 20×5 group. It decreases to -0.806% for a 200×20 group and the average ARPD of all the 12 groups is -0.232%.

Table 8 lists the current best solutions for Tailard benchmark instances. Table 8 also shows that HGA can obtain the best solutions for 115 instances, 92 of which are newly discovered. This is because: (1) LR(n/m) is applied to generate one of individuals in the initial population. (2) Good genes have been selectively inherited across generations. (3) The artificial chromosome maintains the solution stability by participating in crossover directly rather than performing as an individual in the next generation. (4) RZ and FPE are adopted as local search to improve all the offspring of each generation. (5) Dynamic termination criterion is applied.

6. Conclusions

This paper proposes a hybrid genetic algorithm (HGA) for the permutation flowshop scheduling problem (PFSP) with total flowtime minimization. In the HGA, a weighted simple mining gene structure (WSMGS) was established to collect useful information from elite chromosomes and generate an artificial chromosome. A new crossover operator was proposed, which transferred not only local good genes from parents, but also global good genes contained in the artificial chromosome to offspring. A combination of RZ and FPE was applied as local search to improve all generated chromosomes in every generation and LR(n/m) was used for population initialization. A dynamic termination criterion (six consecutive generations without solution improvement) was adopted along with a static termination criterion ($5 \times m \times n$ seconds, rarely satisfied). The HGA outperforms ICH2 and PSO_LS on average and obtains 115 best solutions for 120 Tailard benchmark instances, 92 of which are newly discovered. Local search makes a tremendous contribution to the performance of the HGA. A suitable solution initialization method exerts good influence on the effectiveness and a new crossover operator maintains solution stability for the proposal. Additionally, dynamic termination criterion makes HGA efficient, especially for complex and large problems.

Acknowledgments

This work is supported by the National Natural Science Foundation of China under Grants No. 60672092 and No. 60504029 and the National 863 Project (No. 2008AA04Z103). The authors are grateful to the anonymous referees for their valuable suggestions and comments.

References

Allahverdi, A., Aldowaisan, T., 2002. New heuristics to minimize total completion time in m-machine flowshops. International Journal of Production Economics 77 (1), 71–83.

Chang, P.C., Wang, Y.W., Liu, C.H., 2005. New operators for faster convergence and better solution quality in modified genetic algorithm. Lecture Notes in Computer Science 3611, 983–991.

Chang, Pei-Chann, Chen, Shih-Hsin, Liu, Chen-Hao, 2007. Sub-population genetic algorithm with mining gene structures for multiobjective flowshop scheduling problems. Expert Systems with Applications 33 (3), 762–771.

Framinan, J.M., Leisten, R., 2003. An efficient constructive heuristic for flowtime minimisation in permutation flow shops. Omega 31 (4), 311–317.

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

French, S., 1982. Sequencing and Scheduling an Introduction to the Mathematics of the Job-shop. Ellis Horwood, Chichester.

Gray, M.R., Johnson, D.S., Sethi, R., 1976. The complexity of flowshop and jobshop scheduling. Mathematics of Operations Research 1 (2), 117–129.

Ishibuchi, H., Murata, T., 1998. A multi-objective genetic local search algorithm and its application to flowshop scheduling. IEEE Transactions on Systems Man and Cybernetics Part C – Applications and Reviews 28 (3), 392–403.

Ishibuchi, H., Yoshida, T., Murata, T., 2003. Balance between genetic search and local search in memetic algorithms for multiobjective permutation flowshop scheduling. IEEE Transactions on Evolutionary Computation 7 (2), 204–223.

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

Kim, Yeong-Dae, 1995. Minimizing total tardiness in permutation flowshops. European Journal of Operational Research 85 (3), 541–555.

Li, Xiaoping, Wang, Qian, Wu, Cheng, in press. Efficient composite heuristics for total flowtime minimization in permutation flow shops. Omega (online available).

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

Murata, T., Ishibuchi, H., Tanaka, H., 1996a. Multi-objective genetic algorithm and its applications to flowshop scheduling. Computers and Industrial Engineering 30 (4), 957–968.

Murata, T., Ishibuchi, H., Tanaka, H., 1996b. Genetic algorithms for flowshop scheduling problems. Computers and Industrial Engineering 30 (4), 1061–1071.

Nawaz, M., Enscore, E.E., Ham, I., 1983. A heuristic algorithm for the m-machine n-job flow-shop sequencing problem. Omega 11 (1), 91–95.

Nearchou, A.C., 2004. The effect of various operators on the genetic search for large scheduling problems. International Journal of Production Economics 88, 191–203.

Osman, I.H., Potts, C.N., 1989. Simulated annealing for permutation flow-shop scheduling. Omega 17 (6), 551–557.

Potts, C.N., Van Wassenhove, L.N., 1985. A branch-and-bound algorithm for the total weighted tardiness problem. Operations Research 33 (2), 363–377.

Rajendran, C., 1994. A heuristic for scheduling in flowshop and flowline-based manufacturing cell with multi-criteria. International Journal of Production Research 32, 2541–2558.

Rajendran, C., Ziegler, H., 1997. An efficient heuristic for scheduling in a flowshop to minimize total weighted flowtime of jobs. European Journal of Operational Research 103 (1), 129–138.

Rajendran, C., Ziegler, H., 2004. Ant-colony algorithms for permutation flowshop scheduling to minimize makespan total flowtime of jobs. European Journal of Operation Research 115, 426–438.

Reeves, C.R.A., 1995. genetic algorithms for flowshop sequencing. Computers and Operations Research 22 (1), 5–13.

Ruiz, R., Maroto, C., Alcaraz, J., 2006. Two new robust genetic algorithms for the flowshop scheduling problem. Omega 34, 461–476.

Tailard, E., 1990. Some efficient heuristic methods for the flowshop sequencing problem. European Journal of Operational Research 47 (1), 65–74.

Tailard, E., 1993. Benchmarks for basic scheduling problems. European Journal of Operational Research 64 (1), 278–285.

Tan, Keah-Choon, Narasimhan, Ram, Rubin Paul, A., Ragatz Gary, L., 2000. A comparison of four methods for minimizing total tardiness on a single processor with sequence dependent setup times. Omega 28, 313–326.

Tasgetiren, M. Fatih, Liang, Yun-Chia, Sevkli, Mehmet, Gencyilmaz, Gunes, 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.

Woo, D.S., Yim, H.S., 1998. A heuristic algorithm for mean flowtime objective in flowshop scheduling. Computers and Operations Research 25 (3), 175–182.