Expert Systems with Applications 38 (2011) 7970–7979

Contents lists available at ScienceDirect

Expert Systems with Applications

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

An asynchronous genetic local search algorithm for the permutation flowshop scheduling problem with total flowtime minimization

Xiao Xu, Zhenhao Xu *, Xingsheng Gu

School of Information, East China University of Science and Technology, 200237 Shanghai, PR China

ARTICLE INFO

Keywords:
Scheduling
Genetic algorithm
Permutation flow shop
Total flowtime

ABSTRACT

In this study, the permutation flowshop scheduling problem with the total flowtime criterion is considered. An asynchronous genetic local search algorithm (AGA) is proposed to deal with this problem. The AGA consists of three phases. In the first phase, an individual in the initial population is yielded by an effective constructive heuristic and the others are randomly generated, while in the second phase all pairs of individuals perform the asynchronous evolution (AE) where an enhanced variable neighborhood search (E-VNS) as well as a simple crossover operator is used. A restart mechanism is applied in the last phase. Our experimental results show that the algorithm proposed outperforms several state-of-the-art methods and two recently proposed meta-heuristics in both solution quality and computation time. Moreover, for 120 benchmark instances, AGA obtains 118 best solutions reported in the literature and 83 of which are newly improved.

© 2011 Published by Elsevier Ltd.

1. Introduction

Since Johnson's (1954) pioneering work on the two-machine flowshop, numerous papers on various extensions of this problem have been published in the operation research literature over the last 50 years. The permutation flowshop scheduling problem (PFSP) is a conventional manufacturing system, where n jobs are processed by m machines in the same route (Baker, 1974). Without loss of generality, we can assume that they are processed first on machines 1, then machine 2, etc., until they are processed on machine m. The objective is to find a permutation such that a well-defined measure is minimized. The traditional criterion is the minimization of total completion time, also called makespan. More recently, minimizing total flowtime has become a keen research topic in the scheduling literature. This may be because the criterion is more relevant and meaningful for today's dynamic production environment and is shown to be directly related to the even use of resources, a rapid turn-round of jobs and the minimization of work-in-process inventory (Rajendran & Chaudhuri, 1991).

Exact methods including branch and bound methods (Bansal, 1977; Ignall & Schrage, 1965) and mixed integer linear programming were investigated for solving the PFSP with respect to the total flowtime criterion in early studies. However, in view of the NP-complete nature of the PFSP (Garey, Johnson, & Sethi, 1976), it is unlikely to obtain an optimal solution for the problem in reasonable CPU times.

Hence, many efforts have been devoted to finding nearly optimal or high-quality solutions by using heuristics and meta-heuristics instead of exact methods. Heuristics for the total flow time criterion are generally classified into constructive and composite. The heuristics proposed by Rajendran and Chaudhuri (1991), Rajendran (1993), Wang, Chu, and Proth (1997), Rajendran and Ziegler (1997), Woo and Yim (1998), Liu and Reeves (2001) and Framinan and Leisten (2003) are known to be constructive approaches. Heuristics proposed by Rajendran and Chaudhuri (1991) were based on the minimization of the machine idle times and the job waiting times. Rajendran (1993) then developed a heuristic preference relation to guide the potential job interchanges which improved the heuristic. Wang et al. (1997) addressed two heuristics. The first one was based upon the reduction of machine idle times, while the second one was based upon reducing both the machine idle times and the job queue times. Rajendran and Ziegler (1997) and Liu and Reeves (2001) both presented a kind of sorting method to obtain a seed sequence. The seed sequence is then improved by one round insertion in RZ, while in LR several pairwise exchange schemes are employed to enhance the seed sequence. Woo and Yim (1998) constructed a sequence using an NEH similar method. Framinan and Leisten (2003) also incorporated NEH insertion method as well as the pairwise exchange strategy in their algorithm. At the same time, many composite heuristics have been proposed. Allahverdi and Aldowaisan (2002) investigated seven heuristics which combined with several simple heuristics such as WY and RZ. They demonstrated that the heuristic IH6 outperforms existing heuristics. Subsequently, Framinan and Leisten (2003) proposed IH7-FL algorithm and Framinan, Leisten, and Ruiz-Usano (2005)

* Corresponding author. Tel.: +86 021 64252576.
E-mail address: xuzhenhao@ecust.edu.cn (Z. Xu).

0957-4174/$ - see front matter © 2011 Published by Elsevier Ltd.
doi:10.1016/j.eswa.2010.12.075

addressed C1 and C2 heuristics which integrated with FL and LR method. A comprehensive comparison of existing heuristics was studied by Framinan et al. (2005). They showed that C2 outperforms the existing heuristics. Recently, Li, Wang, and Wu (2009) presented three composite heuristics ICH1–ICH3 where an accelerating scheme named general flowtime computing (GFC) is embedded. They showed that their algorithms are superior to IH7-FL, C1 and C2.

Meta-heuristics commonly have complicated frames, whereas they can always yield better results than constructive and composite heuristics. Consequently, many meta-heuristics were developed to solve the PFSP with the total flowtime criterion. Yamada and Reeves (1998) proposed a genetic algorithm which adopted the stochastic sampling method and the best descent method. Additionally, a multi-step crossover operator is applied in their algorithm. Two ant colony algorithms called M-MMAS and PACO were presented by Rajendran and Ziegler (2004). They showed that their algorithms are clearly superior to the heuristics addressed by Liu and Reeves. In the next year, they slightly improved the two ant colony algorithms and then conducted a comparison study of the four approaches (Rajendran & Ziegler, 2005). Tasgetiren, Liang, Sevkli, and Gencyilmaz (2007) presented a particle swarm optimization algorithm labeled PSO$_{\mathrm{vns}}$ where a SPV rule and VNS local search were applied. The algorithm improved 57 out of the 90 best known solutions reported by Yamada and Reeves (1998) and Rajendran and Ziegler (2004). Subsequently, Jarboui, Ibrahim, Siarry, and Rebai (2008) proposed a combinatorial particle swarm optimization algorithm (H-CPSO) integrated with an improvement procedure. They showed that the H-CPSO get better or equal results comparing with PSO$_{\mathrm{vns}}$. Zhang, Li, and Wang (2009) addressed a hybrid genetic algorithm named HGA. In HGA one individual in the initial population is generated by LR. An artificial chromosome and two heuristics are applied in the algorithm. Their algorithm obtained 115 best results and 92 of which were newly discovered. More recently, Jarboui, Eddaly, and Siarry (2009) presented an estimation of distribution algorithm (EDA), where a variable neighborhood search (VNS) method is embedded. EDA-VNS adopts the probabilistic learning model in global and use VNS to improve the solution quality in local. They showed that both EDA-VNS and VNS yield comparative results comparing with HGA and ICH2. The genetic algorithms presented by Yamada and Reeves (1998) and Zhang et al. (2009) are complicated and time-consuming. However, our genetic algorithm, abbreviated as AGA, is very simple and fast since it does not use the selection and mutation operator which was regarded as an essential part in a conventional genetic algorithm (GA). In the starting phase of AGA, the initial population is generated randomly excepted one gene yielded by a suitable heuristic. In the alteration phase, pairs of chromosomes are randomly selected from current population. The algorithm performs an independent AE strategy in each pair of chromosomes to propagate two new individuals. The offspring population constitute the next generation. The process repeated until the whole individuals' fitness is identical and then a restart scheme is react. Finally, the algorithm terminates when a certain criterion is met. The details of the AE procedure will be described later.

The rest of the paper is structured as follows: Section 2 briefly presented the formulation of PFSP with total flowtime minimization. Proposed algorithm AGA as well as the calibration of the algorithm is described in Section 3. In Section 4, computational results are presented. Finally, Section 5 concludes the paper and suggests some future studies on proposed algorithm.

2. Formulation of the permutation flowshop problem

The PFSP consists of a set of n jobs to be processed on a set of m machines. Each job has m operations and all jobs should be processed on every machine in the same route where the processing time must be fixed and uninterrupted. This problem is often denoted as $n/m/P/\sum C_i$ (Graham, Lawler, Lenstra, & Rinnooy, 1979), where C_j denotes the completion time of job j. Let p_{ij} and $C(j, m)$ denote the processing time and completion time of job j on machine i, respectively. Thus, given a job permutation $\pi = \{\pi_1, \pi_2, \ldots, \pi_n\}$, the completion time of job π_j on machine i can be calculated as follows:

$$\begin{aligned}
&C(1,\pi_1) = p_{1,\pi_1},\\
&C(1,\pi_j) = C(1,\pi_{j-1}) + p_{1,\pi_j}, \quad j = 2,\ldots,n,\\
&C(i,\pi_1) = C(i-1,\pi_1) + p_{i,\pi_1}, \quad i = 2,\ldots,m,\\
&C(i,\pi_j) = max\{C(i-1,\pi_j), \quad C(i,\pi_{j-1})\} + p_{i,\pi_j},\\
&\qquad\qquad i = 2,\ldots,m; \quad j = 2,\ldots,n,
\end{aligned}$$

Since ready times are zero, the flowtime C_{π_j} is equivalent to the completion time $C(\pi_j, m)$. Therefore, the PFSP with the total flowtime criterion is to find a permutation π^* in the set of all permutations $\prod$ such that:

$$\sum_{j=1}^{n} C\left(m,\pi_j^*\right) \le \sum_{j=1}^{n} C(m,\pi_j), \quad \forall \pi \in \prod.$$

3. Proposed algorithm AGA

As aforementioned, since the AE scheme can effectively diversify the population, the selection and mutation operator are not utilized in AGA. Hence, AGA is very simple and easy to implement. The framework of the proposed AE mechanism, simple crossover, E-VNS local search and restart strategy are described thereinafter.

3.1. Representation and population initialization

The well known solution representation for PFSP is the job-based encoding scheme which is often treated in the literature. This scheme is also adopted in AGA since it is very simple.

Randomly generating an initial population is commonly applied in genetic algorithm. The reason is that it may leads to a population diversification which is in favor of the evolution. However, random individuals often have poor fitness and thus may slow down the convergence. To solve the problem, a feasible approach is to employ an effective constructive heuristic. According to Framinan et al. (2005), Zhang et al. (2009) and Li et al. (2009), heuristic LR(n/m) developed by Yamada and Reeves (1998) is an suitable method to obtain fast, good solutions. Hence, in p initial individuals, one sequence is generated by LR(n/m) and the others are yielded randomly.

3.2. Asynchronous evolution

All pairs of individuals were randomly selected from current population and each pair of coupled chromosomes then conducted an AE procedure. Asynchronous evolution mechanism consists of three stages: local search, crossover and local search again.

Given a pair of individuals *father* and *mother* in a certain generation, asynchronous behavior occurred only in the first stage, that is, two parent individuals are improved by different step sizes of local search. More specifically, Let *min-local* and *max-local* denote the minimum and maximum local search step size respectively. Generate a random integer $x1$ in the range (*min-local*, *max-local*) and then *father* carried out $x1$ times of local search. Generate another integer $x2$ and so does the *mother*. The two individuals which have been improved in the first stage then propagate their offspring in the next phase. A very simple crossover operator is employed in this stage, which will be discussed in Section 3.3. In a traditional genetic algorithm, the replacement strategy is that the better offspring will replace their parents, while in AGA a very different update procedure is carried out. The two children (despite worse or better) will conduct *max-local* times of local search before updating. This procedure can ensure that all potentially better sequences will not be immediately abandoned even if they are evaluated as worse genes. The best two individuals of the parents and their children will be retained in the next generation after all these three steps performed.

In the first stage of AE, all individuals change at different speeds. This generates delays in conveying and sharing information among all members of the population. In the meanwhile, the delay creates differences among these individuals in terms of both the fitness and the job sequence, which was regarded as a benefit for getting better results in GA.

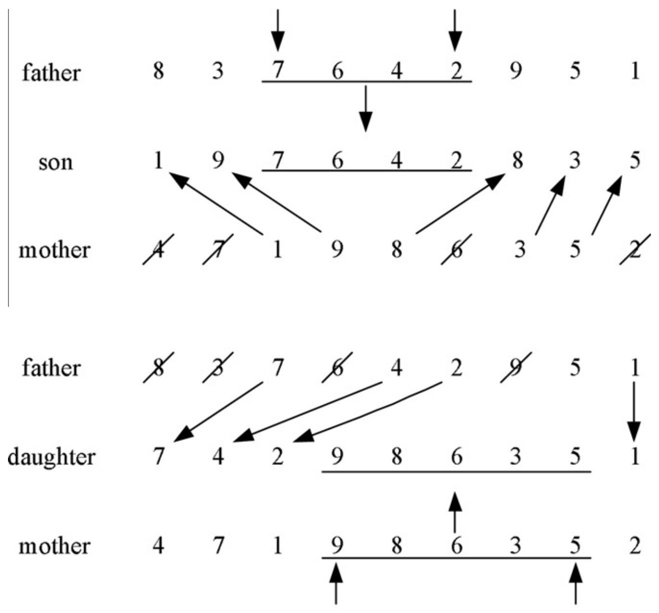

Fig. 1. An example that illustrates the TTP crossover operator.

3.3. Proposed crossover

Generally speaking, the most frequently used crossover operators for permutation-based representations are PMX i.e. partially matched crossover (Goldberg & Lingle, 1985), OX i.e. order-based crossover (Davis, 1985), CX i.e. cycle crossover (Oliver, Smith, & Holland, 1987) and so on. According to Murata, Ishibuchi, and Tanaka (1996), two-point crossover achieved better performance for PFSP problems. As a matter of fact, the two-point crossover version II (TPII) proposed by Murata et al. (1996) is similar to the linear order crossover (LOX). LOX is effective for PFSP since it preserves as much as possible both the relative positions between genes and the absolute position relative to the extremities of the

```
procedure $E-VNS(\pi, power, k)$
        begin
        $loop = 1;$
        while($loop <= k$)
        {
           $flag = 1;$
           while($flag == 1$)
           {
              $flag = 0;$
              for $i = 1: power$ {
                     $x1 \neq x2 = rand(1, n);$
                     $\pi' = insert(\pi, x1, x2);$
                     if($TFT(\pi') \leq TFT(\pi)$) then{ $\pi = \pi'; flag = 1;$ break;} } }
              if($flag == 1$) then {$continue;$}
              else{
                   for $j = 1: power${
                          $x1 \neq x2 = rand(1, n);$
                          $\pi' = exchange(\pi, x1, x2);$
                          if($TFT(\pi') \leq TFT(\pi)$) then{ $\pi = \pi'; flag = 1;$ break;} } }
           }
           $loop++;$
        }
        end
```

Fig. 2. Pseudo code of E-VNS local search.

```
procedure SGL
begin
population = Initialize(P);   // randomly generate P individuals, one provided by LR(n/m)
while(not termination-condition)
{
    while(population not empty)
    {
        father = Random_Select(population);
        mother = Random_Select(population);
        x1, x2 = rand(min_local, max_local);  // randomly draw two intergers between min_local and max_local
        father = E-VNS(father, power, x1);
        mother = E-VNS(mother, power, x2);
        son, daughter = TTP(father, mother);   // applied TTP crossover operator
        E-VNS(son, power, max_local);
        E-VNS(daughter, power, max_local);
        offspring = Add();  // select the best two individuals from parents and children, add them into offspring
    }
    population = offspring;    // population updated
    best permuation = Evaluate(population);
    if(all individuals have inditical fitness) then {
                                        Restart(population); }
}
return best permutation ;
end
```

Fig. 3. Pseudo code description of AGA.

chromosome. Therefore, TP II is served as our crossover operator. TPII crossover only generates one child, so we applied TPII twice in order to obtain two children. Evidently, the two cut points should be different between the two crossing operations. For the sake of simplicity, the proposed crossover can be called twice two-point crossover (TTP). TTP is illustrated in Fig. 1.

For instance, recall the two parent chromosomes *father* and *mother*. First, two crossing points in *father* are randomly chosen, say at position 3 and 6. Then the set of jobs between two cut points are inherited from *father* to *son*. The other jobs in *father* are placed

Table 1
Main results of ANOVA for the experiments on tuning the parameters of AGA.

Source	Sum of squares	Df	F-ratio	p-Value
Main effects				
X1: *p*	0.28939	4	12.10	0.0000
X2: *min-local*	0.02179	3	0.32	0.2827
X3: *max-local*	0.24217	3	8.67	0.0000
X4: *power*	0.08398	4	0.78	0.1328
Interactions				
X1 ∗ *X2*	0.00521	12	0.20	0.66
X1 ∗ *X3*	0.32964	12	9.55	0.0000
X1 ∗ *X4*	0.02424	16	0.16	0.2114
X2 ∗ *X3*	0.26353	9	7.66	0.0032
X2 ∗ *X4*	0.00363	12	0.14	0.7133
X3 ∗ *X4*	0.00255	12	0.1	0.7578

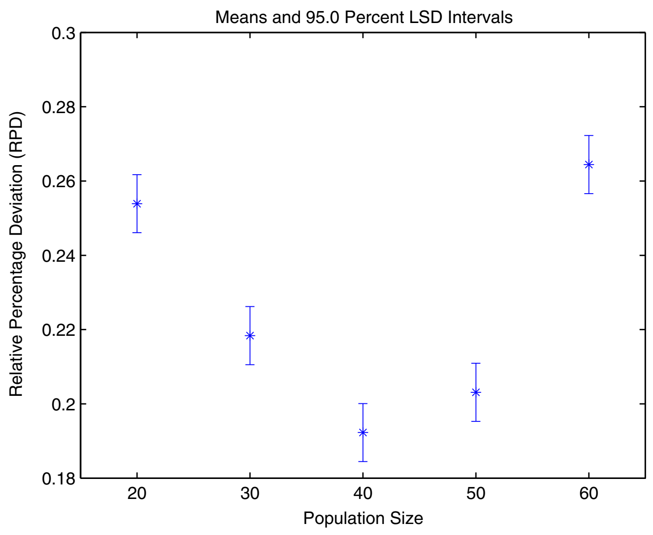

Fig. 4. Means plot and LSD intervals for parameter *p*.

in *son* follow their order in *mother*. Similarly, randomly choose two points in *mother*, say 4 and 8, and then *daughter* is generated by a same procedure.

3.4. Enhanced variable neighborhood search

As mentioned earlier, local search is applied before and after the crossing operation. Local search before crossing is used to diversify the population. In order to make full use of the information of the offspring individuals, local search is used again after the crossing. Therefore, the structure of the local search is rather critical. Two commonly used local search operators, insertion and exchanging operation, are considered. The framework of proposed local search extends the work in Tasgetiren et al. (2007). The pseudo code of the proposed E-VNS local search is given in Fig. 2, where

π	is the sequence needed to be improved
power	is the search times in the inner loop which can be easily adjusted in order to enhance the local search
k	is the searching times in the outer loop which is used to perform AE strategy
$x1$	and $x2$ are the integer numbers randomly generated between 1 and n
insert $(\pi, x1, x2)$	denotes the insertion of job at position $x1$ into position $x2$ in π and
exchange $(\pi, x1, x2)$	denotes the exchanging between the job at position $x1$ and $x2$ in π

Note that the structure unutilized in E-VNS is not identical with which used in VNS developed by Tasgetiren et al. (2007). There are two separate differences between them. On the one hand, the insertion and exchanging operations are strengthened by parameter *power*, that is, in the inner loop, insertion or exchanging operations repeated *power* times until an equal or better sequence is yielded. On the other hand, the search times k in the outer loop is regarded as an input variable of E-VNS instead of a fixed value. The other strategies are similar such as the pivoting rules and the permission of neutral moves. Moreover, Li et al. (2009) presented a method called general flowtime computing (GFC) which can accelerate flowtime computation. In GFC, a schedule is divided into an unchanged subsequence and a changed part. As a result, one just needs to calculate an unchanged part inherited form the parent instead of the complete sequence. GFC is applied in E-VNS since it is rather straightforward.

3.5. Restart mechanism and stopping criterion

Simplifying the genetic scheme may leads to a fast convergence and in the mean while the genes diversity will be lost within short time. To overcome this problem, some restart mechanisms can be incorporated in GA according to Ruiz, Maroto, and Alcaraz (2006). As a result, a very simple restart scheme was developed to renew the population. More specifically, when the fitness of all individuals in the population is identical, a restart scheme then works. The restart scheme implements as follows: first keep the best individual and the seed sequence provided by the heuristic, and then regenerated the remaining individuals randomly.

Regarding the stopping criterion, two commonly used criteria are employed in our proposed algorithm, that is, the maximal computation time and the maximum numbers of generations. AGA terminates whenever either condition is true.

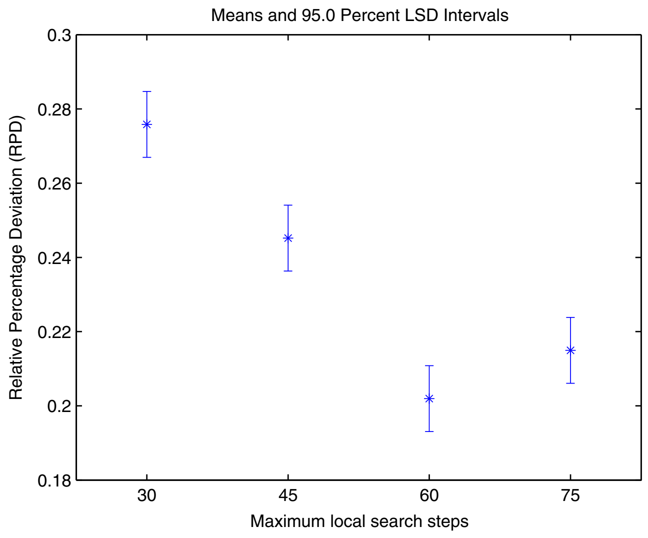

Fig. 6. Means plot and LSD intervals for parameter *max-local*.

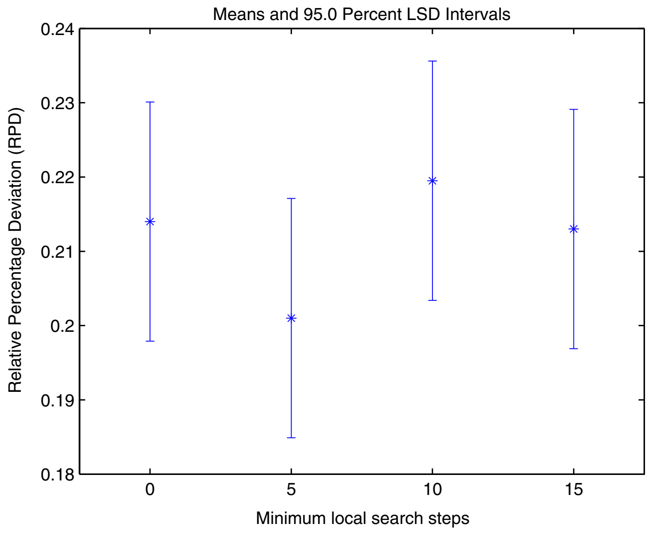

Fig. 5. Means plot and LSD intervals for parameter *min-local*.

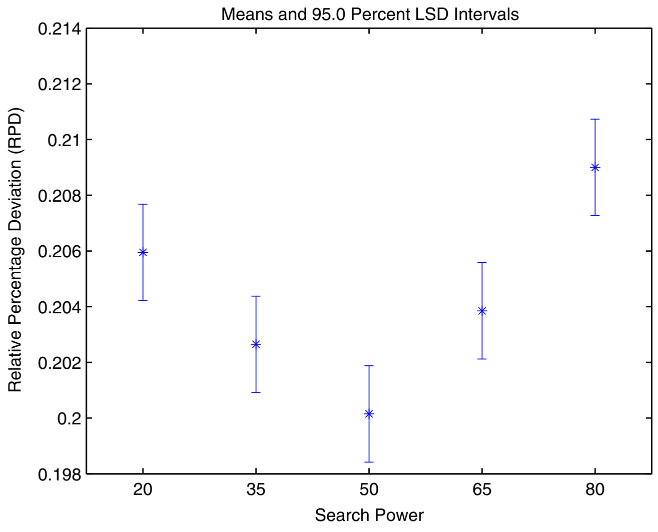

Fig. 7. Means plot and LSD intervals for parameter *power*.

3.6. Outline of the AGA

After the introduction of each module in AGA, we give a pseudo code description of AGA in Fig. 3.

3.7. Calibration of the AGA

Like all meta-heuristics, parameter selection is critical to the AGA performance. AGA has four parameters: the number of population p, the minimum local search step size *min-local*, the maximum local search step size *max-local* and the search power *power*. Although, at present, there is no guidance to select proper values for the parameters, Ruiz and Stützle (2007) presented an effective method to tune parameters, which consist of two procedures called design of experiments (DOE) and multi-factor analysis of variance (ANOVA). Hence, the experiment of parameter determination according to his method is presented as follows:

We first listed the specified levels of all parameters:

- *p:* five levels (20, 30, 40, 50 and 60)
- *min-local:* four levels (0, 5, 10 and 15)
- *max-local:* four levels (30, 45, 60 and 75)
- *power:* five levels (20, 35, 50, 65 and 80)

The above list yields a total of $5 \times 4 \times 4 \times 5 = 400$ different combinations when one carried out a full factorial experimental design. The algorithm is tested with a set of randomly generated PFSP instances. More specifically, we generated 36 different combinations of n and m following the procedure given in Taillard (1993) with $n \in \{20, 30, 40, \ldots, 90, 100\}$ and $m \in \{5, 10, 15, 20\}$. The processing times are distributed uniformly between 1 and 99.

Table 2
Average relative percentage deviation over the best solutions.

Instance	BES(LR)	M-MMAS	PACO	PSO$_{vns}$	H-CPSO	HGA	EDA-VNS	VNS	AGA
20 × 5	1.3610	0.3866	0.6753	0.0632	0.0234	0.0294	0.0000	0.0629	0.0000
20 × 10	1.4330	0.2087	0.5217	0.0465	0.0001	0.0575	0.0075	0.0777	0.0000
20 × 20	1.2242	1.1465	0.9776	3.2012	0.0033	0.0281	0.0000	0.0282	0.0000
50 × 5	1.7256	1.5321	1.3578	0.7923	0.5963	0.2434	0.2436	0.2779	0.2134
50 × 10	2.9071	2.0689	1.8024	1.0656	0.8637	0.3944	0.3120	0.4107	0.3134
50 × 20	2.8946	2.4682	2.1750	2.1479	1.1528	0.8697	0.3354	0.5272	0.3294
100 × 5	1.2673	1.4732	1.5644	0.8232	0.8026	0.8297	0.3071	0.2816	0.2801
100 × 10	2.1627	1.9676	1.6592	1.0246	1.1427	0.9804	0.4869	0.4843	0.4821
100 × 20	3.0656	2.0005	1.8358	1.6004	1.3767	1.1136	0.5543	0.4898	0.4776
Average	1.8041	1.3252	1.2569	1.0865	0.5061	0.4375	0.2247	0.2640	0.2096

Table 3
Peak performance comparison for Taillard instances with $n = 20$. Bold values represent the best solutions for each instance.

Instance	BES(LR)	M-MMAS	PACO	PSO$_{vns}$	H-CPSO	HGA	EDA-VNS/VNS	AGA
20 × 5	14226	14056	14056	**14033**	**14033**	**14033**	**14033**	**14033**
	15446	**15151**	15214	**15151**	**15151**	**15151**	**15151**	**15151**
	13676	13416	13403	**13301**	**13301**	**13301**	**13301**	**13301**
	15750	15486	15505	**15447**	**15447**	**15447**	**15447**	**15447**
	13633	**13529**	**13529**	**13529**	**13529**	**13529**	**13529**	**13529**
	13265	13139	**13123**	**13123**	**13123**	**13123**	**13123**	**13123**
	13774	13559	13674	**13548**	**13548**	**13548**	**13548**	**13548**
	13968	13968	14042	**13948**	**13948**	**13948**	**13948**	**13948**
	14456	14317	14383	**14295**	**14295**	**14295**	**14295**	**14295**
	13036	12968	13021	**12943**	**12943**	**12943**	**12943**	**12943**
Average	1.3610	0.1975	0.4544	0.0000	0.0000	0.0000	0.0000	0.0000
20 × 10	21207	20980	20958	**20911**	**20911**	**20911**	**20911**	**20911**
	22927	**22440**	22591	**22440**	**22440**	**22440**	**22440**	**22440**
	20072	**19833**	19968	**19833**	**19833**	**19833**	**19833**	**19833**
	18857	18724	18769	**18710**	**18710**	**18710**	**18710**	**18710**
	18939	18644	18749	**18641**	**18641**	**18641**	**18641**	**18641**
	19608	**19245**	**19245**	19249	**19245**	**19245**	**19245**	**19245**
	18723	18376	18377	**18363**	**18363**	**18363**	**18363**	**18363**
	20504	**20241**	20377	**20241**	**20241**	**20241**	**20241**	**20241**
	20561	**20330**	**20330**	**20330**	**20330**	**20330**	**20330**	**20330**
	21506	**21320**	21323	**21320**	**21320**	**21320**	**21320**	**21320**
Average	1.4330	0.0492	0.3235	0.0021	0.0000	0.0000	0.0000	0.0000
20 × 20	34119	**33623**	**33623**	34975	**33623**	**33623**	**33623**	**33623**
	31918	31604	31597	32659	**31587**	**31587**	**31587**	**31587**
	34552	**33920**	34130	34594	**33920**	**33920**	**33920**	**33920**
	32159	31698	31753	32716	**31661**	**31661**	**31661**	**31661**
	34990	34593	34642	35455	**34557**	**34557**	**34557**	**34557**
	32734	32637	32594	33530	**32564**	**32564**	**32564**	**32564**
	33449	33038	**32922**	33733	**32922**	**32922**	**32922**	**32922**
	32611	32444	32533	33008	**32412**	**32412**	**32412**	**32412**
	33625	33623	34446	**33600**	**33600**	**33600**	**33600**	**33600**
	32317	32317	33281	**32262**	**32262**	**32262**	**32262**	**32262**
Average	1.0194	0.1189	0.7329	2.2601	0.0000	0.0000	0.0000	0.0000

The stopping criterion is the limited computation time which is fixed at $n \times m \times 0.4$ s and the response variable is calculated by the following equation:

$$\text{Relative percentage deviation (RPD)} = \frac{Obt_{sol} - Best_{sol}}{Best_{sol}} \cdot 100,$$

where Obt_{sol} is the solution yielded by a combination of factors for a given instance and $Best_{sol}$ is the best solution given by all combinations of factors for a instance.

All algorithms throughout the paper were coded in C++ and all tests were conducted on a Pentium IV PC at 2.0 GHz with 1.0 GB memory. Five replicates of each experiment are implemented and the results are analyzed by ANOVA. The three main hypotheses of ANOVA can be easily checked by the means of Ruiz and Stützle (2007). The main results of ANOVA are shown in Table 1.

From Table 1, it is apparent that parameter *p* and *max-local* are very significant. To our surprise, the different levels of *power* does not yield statistically significant differences in the response variable. This may indicates the effectiveness of the proposed restart scheme. Evidently, when the level of *power* is low, the results are not worse since the restart scheme will be frequently triggered. Hence, different levels of *power* are not very statistically significant.

It seems that the role of parameter *min-local* is not important for the algorithm due to its high *p*-value. However, by looking Table 1, we can found that the interaction between *min-local* and *max-local* is very statistically significant. This suggests that the combination of the two parameters is still critical for the AGA performance. As stated earlier, the core mechanism of AGA is the Asynchronous Evolution strategy. More importantly, AE works greatly depending on the length of the interval between *min-local* and *max-local*. Therefore, *min-local* should not be omitted. Thus, all four parameters are further analyzed by multi-compare method using least significant difference (LSD) procedure. The means plot with LSD intervals at a 95% confidence level for the four parameters are shown in Fig. 4–7. As we can see from Fig. 4, populations with a small or large population size do not provide better results, while the difference between *p* = 40 and *p* = 50 are not statistically significant. The means plot for this parameter shown in Fig. 5 testified the conclusion drawn earlier. Different levels of *min-local* seem not critical to the algorithm. As we can see from Fig. 6, a setting of *max-local* = 60 gives a better RPD than a setting of 30 and 45. However, it is not statistically significantly different from 75. A small experiment shows that the results are more promising when the length of the interval between *min-local* and *max-local* takes 50. Therefore, the settings of *min-local* = 10 and *max-local* = 60 are recommended. As for search power, although low level of the parameter dose not generate worse result as seen form Fig. 7, *power* is still set to 50 since we do not want to trigger the restart scheme frequently.

Thus, finally, we fix the parameters *p* = 40 (alternatively *p* = 50), *min-local* = 10, *max-local* = 60 and *power* = 50 in the following experiments.

4. Experimental results

In this section, a comprehensive comparison of AGA with several state-of-the-art methods and two recently well-performing

Table 4
Peak performance comparison for Taillard instances with *n* = 50. Bold values represent the best solutions for each instance. Values with asterisk denote the new best solutions obtained by AGA.

Instance	BES(LR)	M-MMAS	PACO	PSOvns	H-CPSO	HGA	EDA-VNS/VNS	AGA
50 × 5	65663	65768	65546	65058	64838	64956	64817	**64803***
	68664	68828	68485	68298	68223	68298	68066	**68059***
	64378	64166	64149	63577	63436	63513	63240	**63195***
	69795	69113	69359	68571	68590	68571	68287	**68241***
	70841	70331	70154	69698	69584	69562	69478	**69392***
	68084	67563	67664	67138	67062	67111	66882	**66841***
	67186	67014	66600	66338	66375	66318	66274	**66253***
	65582	64863	65123	64638	64531	64418	–	**64365***
	63968	63735	63483	63227	63157	63157	**62981**	**62981**
	70273	70256	69831	69195	69121	69141	**68843**	**68843**
Average	1.7256	1.3039	1.1183	0.4172	0.2920	0.3116	–	0.0000
50 × 10	88770	89599	88942	88031	87672	87593	87238	**87224***
	85600	83612	84549	83624	83199	83517	83116	**82820***
	82456	81655	81338	80609	80311	80136	80132	**80010***
	89356	87924	88014	87053	87037	86953	86725	**86574***
	88482	88826	87801	87263	86819	86905	86626	**86404***
	89602	88394	88269	87255	86735	87008	–	**86723***
	91422	90686	89984	89259	89014	89155	–	**88965***
	89549	88595	88281	87192	87336	87192	87025	**86930***
	88230	86975	86995	86102	85964	86086	85688	**85658***
	90787	89470	89238	88631	88149	88363	–	**87998***
Average	2.9071	1.9085	1.6449	0.6675	0.3427	0.4198	–	0.0000
50 × 20	129095	127348	126962	128622	126126	125850	**125831**	**125831**
	122094	121208	121098	122173	119936	119442	**119247**	119259
	121379	118051	117524	118719	117210	117315	**116696**	**116696**
	124083	123061	122807	123028	121540	121114	120834	**120813***
	122158	119920	119221	121202	118783	118975	118457	**118405***
	124061	122369	122262	123217	120914	120955	120820	**120711***
	126363	125609	125351	125586	123756	123740	123271	**123254***
	126317	124543	124374	125714	122900	122940	122820	**122520***
	125318	124059	123646	124932	122281	122123	**121872**	**121872**
	127823	126582	125767	126311	124529	124936	124486	**124168***
Average	2.9039	1.5825	1.2743	2.1413	0.3686	0.3203	0.0666	0.0010

meta-heuristics is performed. More specifically, AGA is compared with the best method (BES(LR)) extracted from Liu and Reeves (2001); the two ant colony algorithms (M-MMAS and PACO) of Rajendran and Ziegler (2004); the particle swarm optimization algorithm with local search (PSO_{vns}) of Tasgetiren et al. (2007); the combinatorial particle swarm optimization algorithm (H-CPSO) of Jarboui et al. (2008); the hybrid genetic algorithm (HGA) of Zhang et al. (2009); and two recently proposed algorithms (EDA-VNS and VNS) of Jarboui et al. (2009). As aforementioned, all these algorithms as well as AGA were coed in the same programming language and implemented on the same computer. The stopping criteria of these algorithms (i.e. $n \times m \times 0.4$ CPU seconds) are identical. The parameters of AGA are determined according to Section 3.7 and the other algorithms' parameters are set following their papers. Taillard benchmark problems (Taillard, 1993) were adopted to evaluate the algorithm performance. The problem set contains 12 groups of problem instances, each corresponding to a different problem size. There are 10 instances for each given size. The first ninety instances are selected to carry out the comparison experiments. Concerning the performance criterion, RPD defined earlier is still adopted as the evaluation measure, where Obt_{sol} is the solution produced by a given algorithm for an instance and $Best_{sol}$ is the best result obtained by all methods. All algorithms were re-implemented five times. Experimental results are presented in Table 2.

As we can see from Table 2, EDA-VNS, VNS and AGA perform very well for Taillard problems with minor differences between them. AGA outperforms both EDA-VNS and VNS for all cases excepted in 50×10. VNS yields better results than EDA-VNS in relatively large problem size. In addition, in 20×10, AGA has a significantly better performance than these two algorithms. It should be noted that EDA-VNS and AGA have several similar features. For instance, they both employ a variable neighborhood search scheme and they both belong to evolutionary algorithms. Besides, the selection and mutation operator are not applied in both methods. Regarding the average performance, it appears that they are very effective in solving flowshop problems with the total flowtime criterion compared with the existing algorithms. With respect to the rest of the methods, BES(LR) (here we refer to the best composite heuristic reported in Liu and Reeves, 2001) is outperformed by other algorithms. However, BES(LR) is rather simple and easily implemented compared to other algorithms. HGA is superior to M-MMAS, PACO, PSO_{vns} and H-CPSO, although it is relative complicated since it incorporates some complex tricks. Regarding the overall results, AGA dominates the other algorithms in both efficiency and effectiveness.

Although we re-implemented these algorithms honestly and contacted each author to ensure the correctness of the codes, the results given in Table 2 may still did not accurately reflect their performance due to certain reasons. As a consequence, we conducted a peak performance (i.e. the best solutions for the benchmarks published in their papers) comparison among all these algorithms. The results are shown in Tables 3–6.

As can be seen from the three tables, H-CPSO, HGA, EDA-VNS/VNS and AGA reach all the best solutions for the small problem size, whereas, for the remaining instances, AGA produced 83 out of 90 new best solutions. In addition, AGA improved all best solutions provided by HGA excepted one instance. The RPD of HGA is greatly reduced by the proposed algorithm. Hence, obviously, AGA has a profound performance by comparing with these

Table 5
Peak performance comparison for Taillard instances with $n = 100$. Bold values represent the best solutions for each instance. Values with asterisk denote the new best solutions obtained by AGA.

Instance	BES(LR)	M-MMAS	PACO	PSO_{vns}	H-CPSO	HGA	EDA-VNS/VNS	AGA
100×5	256789	257025	257886	254762	255520	254762	254250	**253926***
	245609	246612	246326	245315	244511	243850	243227	**242886***
	241013	240537	241271	239777	239843	239173	238580	**238280***
	231365	230480	230376	228872	229481	228705	228520	**228169***
	244016	243013	243457	242245	242229	241432	241397	**240810***
	235793	236225	236409	234082	234394	233698	233161	**232876***
	243741	243935	243854	242122	242779	241650	241213	**240951***
	235171	234813	234579	232755	232889	232734	231865	**231804***
	251291	252384	253325	249959	250294	249920	249038	**248679***
	247491	246261	246750	244275	244903	244131	243647	**243518***
Average	1.2673	1.2220	1.3412	0.5101	0.6214	0.3389	0.1246	0.0000
100×10	306375	305004	305376	303142	302971	300507	301001	**300473***
	280928	279094	278921	277109	277408	277109	275601	**275298***
	296927	297177	294239	292465	291669	290468	288943	**288707***
	309607	306994	306739	304676	305663	303443	–	**302635***
	291731	290493	289676	288242	287761	286647	–	**285643***
	276751	276449	275932	272790	274152	272764	271956	**271475***
	288199	286545	284846	282440	282602	282373	281090	**281057***
	296130	297454	297400	293572	295430	293067	–	**292471***
	312175	309664	307043	305605	305819	304330	303893	**303829***
	298901	296869	297182	295173	296425	293932	293492	**293159***
Average	2.1733	1.7633	1.4737	0.7056	0.8682	0.3473	–	0.0000
100×20	383865	373756	372630	374351	372480	369652	368641	**368590***
	383976	383614	381124	379792	376476	376067	374838	**374086***
	383779	380112	379135	378174	375733	373199	372423	**372142***
	384854	380201	380765	380899	379273	374832	–	**374601***
	383802	377268	379064	376187	374416	371268	–	**370752***
	387962	381510	380464	379248	378380	375463	375348	**374871***
	384839	381963	382015	380912	379395	376353	–	**376350***
	397264	393617	393075	392315	390587	387189	–	**387019***
	387831	385478	380359	382212	380762	378657	377729	**377712***
	394861	387948	388060	386013	384734	382004	381623	**380971***
Average	3.0892	1.8197	1.5868	1.4119	0.9357	0.2027	–	0.0000

Table 6
Peak performance comparison for Taillard instances with $n = 100$ and $n = 500$. Bold values represent the best solutions for each instance. Values with asterisk denote the new best solutions obtained by AGA.

Instance	BES(LR)	M-MMAS	PACO	PSO_{vns}	H-CPSO	HGA	EDA-VNS/VNS	AGA
200 × 10	1063976	–	–	–	–	1050564	–	**1049830***
	1049076	–	–	–	–	1040604	–	**1036427***
	1059765	–	–	–	–	1050785	–	**1048993***
	1051335	–	–	–	–	1038885	–	**1033110***
	1055823	–	–	–	–	1041510	–	**1038288***
	1023054	–	–	–	–	1013421	–	**1011864***
	1071471	–	–	–	–	1061285	–	**1059727***
	1054500	–	–	–	–	1049007	–	**1048299***
	1045183	–	–	–	–	1026991	–	**1026137***
	1044888	–	–	–	–	1038016	–	**1035409***
Average	1.2625	–	–	–	–	0.2117	–	0.0000
200 × 20	1247352	–	–	–	–	1235238	–	**1234223***
	1271603	–	–	–	–	1254529	–	**1253715***
	1297768	–	–	–	–	1274798	–	**1273570***
	1272199	–	–	–	–	1245656	–	**1243223***
	1255708	–	–	–	–	1236246	–	**1231608***
	1251817	–	–	–	–	1237754	–	**1231235***
	1275658	–	–	–	–	1248821	–	**1248109***
	1273142	–	–	–	–	1249644	–	**1248110***
	1259311	–	–	–	–	1237428	–	**1237168***
	1273354	–	–	–	–	1253075	–	**1250596***
Average	1.8172	–	–	–	–	0.1745	–	0.0000
500 × 20	6746310	–	–	–	–	6723143	–	**6718965***
	6868018	–	–	–	–	6844840	–	**6841013***
	6793698	–	–	–	–	6772110	–	**6743171***
	6812857	–	–	–	–	6809460	–	**6802933***
	6767964	–	–	–	–	6742209	–	**6737370***
	6774423	–	–	–	–	**6729388**	–	6738575
	6739792	–	–	–	–	6706950	–	**6691468***
	6821619	–	–	–	–	6795769	–	**6790270***
	6753839	–	–	–	–	6736573	–	**6715549***
	6778403	–	–	–	–	6764295	–	**6760926***
Average	0.4833	–	–	–	–	0.1390	–	0.0137

Table 7
Comparison of AGA with and without AE in terms of RPD within a given computation time.

Instance	20 × 5	20 × 10	20 × 20	50 × 5	50 × 10	50 × 20	100 × 5	100 × 10	100 × 20	Average
AGA_AE	0.0000	0.0000	0.0000	0.2184	0.3057	0.2494	0.2617	0.4721	0.4666	0.2093
30	0.0111	0.1071	0.0103	0.4493	0.8130	1.4135	0.9213	1.2303	1.3441	0.7000
45	0.0179	0.1019	0.0076	0.4305	0.7932	1.4179	0.9131	1.1289	1.2321	0.6715
60	0.0313	0.0319	0.0113	0.4413	0.8057	1.3931	0.9204	1.1107	1.2431	0.6654
75	0.0397	0.1311	0.0098	0.4469	0.8025	1.4032	0.9168	1.2259	1.3593	0.7039

competence meta-heuristics. As mentioned earlier, the effectiveness of AGA may be because of the incorporation of the AE scheme. In order to testify the supposition, a comparison between AGA with AE scheme and AGA without AE scheme was carried out. AGA without AE means that local search time before crossing is fixed to certain values which are listed in the first column of Table 7. The experimental results are given in Table 7. From Table 7, we can observe that AGA with AE is clearly superior to AGA with fixed local search time. The results suggest that asynchronous evolution scheme is rather effective for AGA.

5. Conclusions and future works

In this paper, we presented a novel genetic algorithm called asynchronous genetic local search algorithm. In AGA, one solution was yielded by an effective constructive heuristic and the remaining members were generated randomly. The initial population then conducted a simple crossover operator which was slightly modified by the two-point crossover. An enhanced variable neighborhood search named E-VNS was applied before and after the crossover operation. Different times of local search before crossing are regarded as an asynchronous evolution behavior. The experimental results showed that AE is effective for AGA. Additionally, an easily implementable restart scheme was developed to avoid the premature of the algorithm since the selection and mutation strategies are not utilized.

To evaluate the performance of AGA, comparison tests with certain state-of-the-art methods and two recently proposed meta-heuristics are carried out. The results show that AGA outperforms the existing algorithms in terms of the solution quality and computation time. Ultimately, 83 out of 90 best solutions provided by HGA, EDA-VNS and VNS were improved by AGA. For future research, a more sophisticated restart mechanism should be further studied. More effective restart scheme may greatly improve the algorithm performance. In addition, a very small experiment shows that AGA can obtain promising results for the flowshop problem with respect to makespan criterion. Thus, further studies on AGA may focus on the PFSP with makespan minimization.

Acknowledgements

This work is funded by the National Natural Science Foundation of China (No. 60774078), the National High Technology Research and Development Program of China (No. 2009AA04Z141), Shanghai Municipal Natural Science Foundation (No. 10ZR1408300) and the Fundamental Research Funds for the Central Universities. The authors would also like to thank Dr. Rubén Ruiz for his constructive comments and contribution to the completion of the work.

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.

Baker, K. R. (1974). *Introduction to sequencing and scheduling*. New York: Wiley.

Bansal, S. P. (1977). Minimizing the sum of completion times of n-jobs over m-machines in a flowshop – A branch and bound approach. *AIIE Transactions, 9*, 306–311.

Davis, L. (1985). Applying adaptive algorithms to epistatic domains. In *Proceedings of the international joint conference on artificial intelligence* (pp. 162–164).

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

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

Garey, M. R., Johnson, D. S., & Sethi, R. (1976). The complexity of flow shop and jobshop scheduling. *Mathematics of Operations Research, 2*, 117–129.

Goldberg, D., & Lingle, R. Jr., (1985). Alleles, loci, and the traveling salesman problem. In *Proceedings of first international conference on genetic algorithms and their applications* (pp. 154–159). Hillsadale, NJ: Lawrence Erlbaum Associates.

Graham, R. L., Lawler, F. L., Lenstra, J. K., & Rinnooy, K. A. (1979). Optimization and approximation in deterministic sequencing and scheduling: A survey. *Annals of Discrete Mathematics, 5*, 287–326.

Ignall, E., & Schrage, L. (1965). Application of the branch and bound technique to some flowshop scheduling problem. *Operations Research, 13*, 400–412.

Jarboui, B., Eddaly, M., & Siarry, P. (2009). An estimation of distribution algorithm for minimizing the total flowtime in permutation flowshop scheduling problems. *Computers and Operations Research, 36*, 2638–2646.

Jarboui, B., Ibrahim, S., Siarry, P., & Rebai, A. (2008). A combinatorial particle swarm optimization for solving permutation flowshop problems. *Computers and Industrial Engineering, 54*, 526–538.

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

Li, Xiaoping, Wang, Qian, & Wu, Cheng (2009). Efficient composite heuristics for total flowtime minimization in permutation flow shops. *Omega, 37*, 155–164.

Liu, J., & Reeves, C. R. (2001). Constructive and composite heuristics solution of the scheduling problem. *European Journal of Operational Research, 132*, 439–452.

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

Oliver, I. M., Smith, D. J., & Holland, J. R. C. (1987). A study of permutation crossover operators on the traveling salesman problem. In *Proceedings of the second international conference on genetic algorithms and their applications, Hillsadale, NJ* (pp. 224–230).

Rajendran, C. (1993). Heuristic algorithm for scheduling in flow shop to minimize total flow time. *International Journal of Production Economics, 29*, 65–73.

Rajendran, C., & Chaudhuri, D. (1991). An efficient heuristic approach to the scheduling of jobs in a flow shop. *European Journal of Operation Research, 61*, 318–325.

Rajendran, C., & Ziegler, H. (1997). An efficient heuristic for scheduling in a flow shop to minimize total weighted flowtime of jobs. *European Journal of 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. *Computers and Industrial Engineering, 48*, 789–797.

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

Ruiz, R., & Stützle, T. (2007). A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. *European Journal of Operational Research, 177*, 2033–2049.

Taillard, E. (1993). Benchmarks for basic scheduling problems. *European Journal of Operational Research, 64*(2), 278–285.

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.

Wang, C., Chu, C., & Proth, J. M. (1997). Heuristics approaches for scheduling problems. *European Journal of Operational Research, 96*, 636–644.

Woo, D. S., & Yim, H. S. (1998). A heuristic algorithm for mean flow time objective in flow shop scheduling. *Computers and Operations 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 1998 IEEE international conference on evolutionary computation* (pp. 230–234).

Zhang, Y., Li, Xiaoping, & Wang, Qian (2009). Hybrid genetic algorithm for permutation flowshop scheduling problems with total flowtime minimization. *European Journal of Operational Research, 196*, 869–876.