Effective metaheuristics for scheduling a hybrid flowshop with sequence-dependent setup times

Quan-Ke Pan^a, Liang Gao^{a*}, Xin-Yu Li^a, Kai-Zhou Gao^b

State Key Lab of Digital Manufacturing Equipment & Technology in Huazhong University of Science &
Technology, Wuhan, 430074, P. R. China

School of Computer Science, Liaocheng University, Liaocheng, 252000, P. R. China

Abstract: This paper proposes a total of nine algorithms to minimize the makespan for the hybrid flowshop scheduling problem with sequence-dependent setup times. The first six algorithms are trajectory-based metaheuristics, including three variants of iterated local search and three variants of iterated greedy. The remaining three algorithms are population-based metaheuristics, namely, the improved fruit fly optimization, the improved migrating birds optimization, and the discrete artificial bee colony optimization. We present some advanced and effective technologies, including three mixed neighborhood structures, an enhanced perturbation method, and an enhanced destruction and construction procedure for the trajectory-based metaheuristics. We propose a path-relinking-based cooperative search, a diversity control scheme, and a diversified initialization approach for the improved fruit fly optimization. We calibrate the parameters and operators for the proposed algorithms by means of a design of experiments approach. To evaluate the proposed algorithms, we present several adaptations of other recent well-known meta-heuristics for the problem and conduct a comprehensive set of computational and statistical experiments to demonstrate the effectiveness of the presented algorithms. Among them, the discrete artificial bee colony optimization is the best-performing algorithm and it is able to improve 126 out of the 240 best known solutions for the benchmarks in the literature.

Keywords: Scheduling; Flowshop; Iterated local search; Iterated greedy algorithm; Fruit fly optimization; Migrating birds optimization; Artificial bee colony;

* Corresponding author. Email: Gaoliang@mail.hust.edu.cn. Tel: +86 27 87544522

1. Introduction

Scheduling is a decision-making process for committing resources to a variety of possible tasks. Effective scheduling methods and technologies have become essential for industries to maintain competitive positions in today's fast-changing business environment [1]. Hybrid flowshop scheduling (HFS) is a well-studied machine scheduling problem that arises in applications in the chemical, textile, glass, and many other industries. HFS simulates common manufacturing processes consisting of a set of production stages, and at least one stage has two or more identical parallel machines. It can be considered as an extension of the regular flowshop scheduling problem. Unlike the regular flowshop scheduling problem, however, where a permutation of jobs for each machine is the only thing needed, HFS, with potentially more than one machine per stage, has to consider the assignment of jobs to machines and the sequence of jobs on each machine [2].

On many production floors, setup time is a significant factor for production scheduling. Pinedo [1] noted that the improper handling of setup operations may easily consume more than 20% of the available machine capacity. Setup operations include cleaning, fixing or releasing parts to machines, changing tools, and adjustments to machines. They are non-productive operations performed on machines to prepare them for the next product. Setup times are not part of the processing times and are often separable from them, *i.e.*, setup operations can be performed before the product arrives at the machine. Setup times are often strongly dependent on the stage, on the job that the machine is processing, and on the job that comes next. Therefore, it is practical and realistic to integrate sequence-dependent setup times (SDSTs) into HFS.

SDSTs significantly increase the difficulty of scheduling problems. This paper addresses the objective of minimizing the makespan, that is, the maximum completion time across all jobs, for the HFS problem with SDST (SDST/HFS for short). The criterion tends to a high throughput and a high utilization of production resources, and it is widely used in the scheduling literature. SDST/HFS with the makespan criterion is denoted as $FHm/S_{ijk}/Cmax$ using the well-known $\alpha/\beta/\gamma$ notation [3] and its extension for HFS problems [4]. The HFS scheduling problem was proven by Gupta [5] to be NP-hard even if it only has two stages ($m=2$) and one of the stages contains a single machine, so the SDST/HFS problem is strongly NP-hard as well. Therefore, for several decades, efforts have been dedicated to finding high-quality solutions in a reasonable computational time by heuristic and meta-heuristic optimization techniques instead of finding an optimal solution.

We review some works that dealt with problems related to the problem considered in this paper. Kurz and Askin [6] presented three heuristics based on simple greedy methods, insertion heuristics and the adaptation of Johnson's rule for a SDST/HFS. Later, the same authors [7] formulated the problem as an integer programming model and developed a random key genetic algorithm (GA). By imitating the defence process of the immune system against invaders in a biological body, Zandieh *et al.* [8]

proposed an immune algorithm and established by computational comparison that the proposed immune algorithm performed much better than the GA of Kurz and Askin [7]. Gómez-Gasquet *et al.* [9] presented an agent-based genetic algorithm for a SDST/HFS to minimize the makespan.

Solution techniques are also provided for SDST/HFS with more complex settings. Ruiz and Maroto [10] proposed a GA to find the minimum makespan for a SDST/HFS problem with unrelated parallel machines and machine eligibility. Naderi *et al.* [11] proposed a dynamic dispatching rule heuristic and an iterated local search (ILSN for short, where the capital letter N is taken from the authors' name) to solve a SDST/HFS problem with machine eligibility and stage skipping. Naderi *et al.* [12] introduced an improved simulated annealing (ISA) for a SDST/HFS problem involving transportation times to minimize the total completion time and total tardiness. Ruiz *et al.* [13] considered a mixed integer programming model and some heuristics for a realistic HFS problem with unrelated machines, setup times, overlaps, precedence relationships and other parameters. Jungwattanakit *et al.* [14] proposed an integer programming model, dispatching rules and a GA for a SDST/HFS problem to minimize the weighted sum of the makespan and the number of tardy jobs. The same authors later investigated the same problem using simulated annealing and tabu search methods [15]. Recently, Ebrahimi *et al.* [16] presented two meta-heuristics based on the non-dominated sorting GA [17] and the multi-objective GA [18] for a HFS problem with sequence-dependent family setup times. Two comprehensive reviews on HFS problems can be found in [19] and [20].

The iterated local search (ILS) metaheuristic presented by Lourenço *et al.* (2003) [21] and the iterated greedy (IG) introduced by Ruiz and Stützle [22] are two trajectory-based metaheuristics that have resulted in top-notch performance for many scheduling problems despite their simplicity [23-29]. Fruit fly optimization (FFO) [30], migrating birds optimization (MBO) [31], and artificial bee colony (ABC) [32] are three recently presented population-based metaheuristics that have already been successfully applied to many scheduling problems, including the semiconductor final testing scheduling problem [33], the HFS problem [34, 35], the permutation flowshop problem [36], the order acceptance and scheduling problem [37], and steel-making and continuous casting scheduling [38]. Due to the effectiveness of the above metaheuristics, it is worthwhile to evaluate them for the SDST/HFS problem, an NP-hard combinatorial optimization problem with a huge solution space.

In this paper, our aim is to tackle the SDST/HFS problem with the objective of minimizing the makespan. Based on the five abovementioned metaheuristics, we propose a total of nine algorithms, including three simple variants of iterated local search, **three simple variants of iterated greedy, the improved** fruit fly optimization, the improved migrating birds optimization, and the discrete artificial bee colony optimization. We introduce three mixed neighborhood structures, an enhanced perturbation method, an enhanced destruction and construction procedure, a path-relinking-based cooperative search, a diversity control scheme, and a diversified initialization approach to stress the performance of the proposed algorithms.

To our best knowledge, this is the first reported application of iterated greedy, fruit fly optimization, migrating birds optimization, and artificial bee colony for the considered SDST/HFS problem. To investigate the effectiveness of the parameters and operators for each of the nine algorithms, a design of experiments (DOE) approach and analysis of variance (ANOVA) are utilized. The effectiveness of the proposed algorithms is demonstrated by extensive comparisons against the existing methods. In addition, 126 out of the 240 best known solutions for the instances presented in [39] are further improved by the proposed algorithms.

The rest of the paper is organized as follows. In Section 2, SDST/HFS with the makespan criterion is formulated. Sections 3 and 4 present six trajectory-based metaheuristics and three population-based metaheuristics in detail, respectively. In Section 5, we calibrate the presented algorithms, and we report the computational results and comparisons in Section 6. Finally, Section 7 provides the concluding remarks and suggests some future work.

2. The SDST/HFS problem

The SDST/HFS problem can be described as follows. A set of n jobs $J = \{1, 2, \dots, n\}$ have to be processed through m production stages $\{1, 2, \dots, m\}$ following the same production route, *i.e.*, first at stage 1, then at stage 2, and so on until last stage m . Each stage k , $k = 1, 2, \dots, m$, has a set of identical parallel machines, M_k ($|M_k| \geq 2$ for at least one stage, where $|\cdot|$ denotes the cardinality of a set). Each job $j \in J$ can be processed on any one of the $|M_k|$ machines at stage k and the processing time is the same on each machine. We denote the processing time of job $j \in J$ at stage k as $p_{k,j}$. We have an SDST, denoted as $s_{k,j',j}$, when job $j \in J$ is processed immediately after job $j' \in J$ ($j' \neq j$) on the same machine at stage k . If job $j \in J$ is the first job processed on a machine at stage k , its setup time is denoted as $s_{k,j,j}$. At any time, no job can be processed on more than one machine, and no machine can process more than one job. All jobs are independent and available for processing at time 0. The objective is then to find a schedule so that the makespan (denoted by $C_{\max}$) is minimized.

For an example, we consider a problem with four jobs ($n = 4$) and two stages ($m = 2$), with two machines at stage one ($|M_1| = 2$) and three machines at stage two ($|M_2| = 3$). The processing times and setup times are given in Tables 1 and 2, respectively. A schedule chart is shown in Fig. 1.

[Insert Table 1 here]

[Insert Table 2 here]

[Insert Fig. 1 here]

3. Proposed trajectory-based metaheuristics

Trajectory-based metaheuristics manipulate only one candidate solution of the given problem instance in each search step. ILS and IG are two well-known trajectory-based metaheuristics. This section presents three variants of ILS and three variants of IG for the SDST/HFS with the makespan criterion. We detail the solution representation, the local search methods, the proposed ILS algorithms, and the proposed IG algorithms as follows.

3.1. Solution representation

Permutation-based encoding is very simple and it is widely used in the HFS problem literature [10], so we choose this representation for our metaheuristics. In the permutation-based encoding, a permutation of all jobs represents the order in which the jobs are launched into the shop at the first stage. In the subsequent stage, the completion times of the jobs in the previous stage are taken as their release times. The jobs are allocated to the first available machine according to the First In First Out rule. In the regular HFS problem, the first available machine also results in the earliest completion time for a job. However, this is not the case for the SDST/HFS problem because SDSTs are involved. The first available machine might need a large SDST, so assigning the job to this machine would result in a later completion time than to the other machines. To solve this problem, we assign a job to the machine that can finish it at the earliest time. For a job permutation $\pi = (\pi_1, \pi_2, \dots, \pi_n)$, $\pi_j \in J$, $j = 1, 2, \dots, n$, we consider the machine assignment and completion time of π_j as follows.

At stage k , job π_j can be assigned to any one of the $|M_k|$ machines. For machine $i \in M_k$ that has not previously processed any job, if we assign job π_j to it, the completion time of job π_j on machine i at stage k , $C_{k,\pi_j,i}$, can be calculated by

$$C_{k,\pi_j,i} = \max\{s_{k,\pi_j,\pi_j}, C_{k-1,\pi_j}\} + p_{k,\pi_j} \quad (1)$$

If machine i has already processed some jobs, we have

$$C_{k,\pi_j,i} = \max\{C_{k,\tau_{k,i}} + s_{k,\tau_{k,i},\pi_j}, C_{k-1,\pi_j}\} + p_{k,\pi_j} \quad (2)$$

where $\tau_{k,i}$ represents the last job that machine i has already processed. Finally, we assign job

π_j to machine $i^* = \arg\left(\min_{i=1,2,\dots,|M_k|} \{C_{k,\pi_j,i}\}\right)$ that leads to the earliest completion time. The completion

time of job π_j at stage k , C_{k,π_j} , is given by

$$C_{k,\pi_j} = \min_{i=1,2,\dots,M_k} \{C_{k,\pi_j,i}\} \quad (3)$$

Once all jobs are assigned to machines at all stages, we obtain the makespan by

$$C_{\max} = \max_{j=1,\dots,n} \{C_{m,\pi_j}\} \quad (4)$$

3.2. Local search methods

The performance of local search methods depends significantly on the underlying neighborhood structure. Two commonly used neighborhoods for the job permutation based representation are generated by insertion and interchange. Insertion extracts a job from the permutation and inserts it into a different position, whereas interchange chooses two jobs and interchanges their positions. Insertion and interchange lead to distinct neighborhood structures for the SDST/HFS problem. We consider three different local search methods based on the insertion and interchange for our trajectory-based algorithms.

3.2.1 Hybrid local search

We first consider a simple hybridization of the insertion and interchange operators. A random number $rand(\)$ is generated in the range of $[0, 1]$ with a uniform distribution. If $rand(\)$ is less than β , a control parameter, then a neighboring solution is randomly selected in the insertion neighborhood; otherwise, we randomly choose a neighboring solution from the interchange neighborhood. We denote this hybrid local search procedure as HLS. Let π be a solution, and $loop_{HLS}$ the maximum number of neighbors to be explored. The pseudo codes of HLS is

[Insert Fig. 2 here]

3.2.2 Variable neighborhood search

The second local search procedure is a simple variant of the traditional variable neighborhood search (VNS) proposed by Mladenovic and Hansen [40]. Unlike the traditional VNS which explores a smaller neighborhood first and then moves to a bigger one, our method just explores two quite different neighborhoods, *i.e.*, the insertion and interchange neighborhoods. Following Tasgetiren *et al.* (2007) [41], Naderi and Ruiz (2010) [42], and Tasgetiren *et al.* (2011) [43], our method randomly finds a neighboring solution for the current incumbent solution in its insertion neighborhood. If an improvement is made, then the neighboring solution becomes the new incumbent solution, and the

insertion neighborhood is explored again; otherwise, the procedure randomly searches for a neighboring solution for the current incumbent solution in its interchange neighborhood. Different from the HLS, the VNS prefers the insertion neighborhood and an insertion operator is always performed if an improvement is made in the search process. The pseudo codes of the VNS is given in Fig. 3, where $loop_{VNS}$ is a control parameter.

[Insert Fig. 3 here]

3.2.3 Mixed referenced local search

In our previous study [44, 26, 27], a referenced local search for the insertion neighborhood was successfully applied to the permutation flowshop problem, the lot-streaming flowshop problem, and the mixed no-idle flowshop problem. The referenced local search sequentially extracts a job from the current sequence π according to a seed sequence π^s and reinserts it at all possible positions. The best position of the insertion for the job is identified, and the resulting sequence is used to replace the current sequence if there is an improvement in the makespan value. The referenced local search is repeated until no better solution is found. The referenced local search with insertion, denoted as RLS_{insert} , is described in Fig. 4.

[Insert Fig. 4 here]

Similarly, we present a referenced local search based on the interchange operator, denoted as $RLS_{Interchange}$. To avoid a cyclic search, the current job is only interchanged with the jobs before it is subjected to the permutation (Dong *et al.* 2015) [24]. Suppose $(\pi_1, \pi_2, \dots, \pi_{i-1}, \pi_i, \pi_{i+1}, \dots, \pi_n)$ is the current permutation and π_i is the current job. Only job π_j , $j=1, 2, \dots, i-1$, can be interchanged with π_i in our $RLS_{Interchange}$. The procedure of $RLS_{Interchange}$ is given in Fig. 5.

[Insert Fig. 5 here]

Based on RLS_{insert} and $RLS_{Interchange}$, we introduce a mixed referenced local search (MRLS) for the SDST/HFS problem under consideration. Let α be a control parameter that is used to determine which local search is to be performed. The procedure of MRLS is given in Fig. 6.

[Insert Fig. 6 here]

3.3 Iterated Local search

ILS is a simple but powerful optimizer that consists of initialization, a local search, perturbation, and an acceptance criterion. We use the adaption to the SDST/HFS of the NEH heuristic (Nawaz *et al.*, 1983) [45] to generate an initial solution for the presented ILS algorithm. NEH is an effective heuristic for the permutation flowshop problem with the makespan criterion. It generates a seed sequence using the largest processing time rule. The first job in the seed sequence is taken as the first job of the current sequence, and the remaining jobs are extracted one by one from the seed sequence and then inserted into the best slot of the current sequence.

The perturbation procedure consists of a number ω of random insertions to the current solution. A random perturbation procedure will lead to, most likely, a poor solution, which would be a waste of time. To increase the chance of ILS moving to a more promising region, as performed by Naderi *et al.* (2010) [11], we generate a number ϖ of new solutions by performing the perturbation procedure on the current solution ϖ times. All of these ϖ new solutions are evaluated, and the best one is chosen for our ILS. The perturbation procedure is given in Fig. 7.

[Insert Fig. 7 here]

For the acceptance criterion, we consider the simulated annealing type criterion with a constant temperature T [46, 23]. The temperature T depends on the particular instance, and it is computed as

$$T = \lambda \cdot \frac{\sum_{j=1}^n \sum_{k=1}^m P_{k,j}}{10nm} \quad (5)$$

where λ is a parameter that needs to be adjusted. The acceptance criterion is given in Fig. 8.

[Insert Fig. 8 here]

With the above design for the initialization, perturbation, acceptance criterion and local methods, we present three simple ILS variants for the considered SDST/HFS problem: ILS with HLS (ILS_{HLS}), ILS with VNS (ILS_{VNS}), and ILS with MRLS (ILS_{MRLS}). Let π^* be the best solution found so far. The procedures of the three presented ILS variants are given in Fig. 9, Fig. 10, and Fig. 11.

[Insert Fig. 9 here]

[Insert Fig. 10 here]

[Insert Fig. 11 here]

3.4 Iterated Greedy

IG was introduced by Ruiz and Stützle [22] for permutation flowshop with the makespan criterion. IG is the same as ILS except that it has a destruction and construction (DC) procedure instead of a perturbation operator. During the destruction phase, a number d of jobs are randomly removed from the current solution. Afterwards, the construction procedure reinserts the removed d jobs one by one into the best slot of the current sequence. Our previous study [26] presented an enhanced DC procedure for the mixed no-idle flowshop problem. In the enhanced DC procedure, after reinserting one job, the jobs occupying the previous and posterior positions are also reinserted by considering all of the possible slots in the current solutions. We combine the basic and enhanced DC procedures and obtain a mixed DC (MDC) procedure. In the MDC, the parameter Enh , decides which DC procedure is used. The MDC procedure is given in Fig. 12.

[Insert Fig. 12 here]

We present three simple IG variants for the considered SDST/HFS problem: the IG with HLS (IG_{HLS}), the IG with VNS (IG_{VNS}), and the IG with MRLS (IG_{MRLS}). The procedures of the three presented IG variants are separately given in Fig. 13, Fig. 14, and Fig. 15.

[Insert Fig. 13 here]

[Insert Fig. 14 here]

[Insert Fig. 15 here]

4. Population-based Metaheuristics

In population-based metaheuristics, several candidate solutions of the given problem instance are simultaneously maintained at each search step. We present three population-based metaheuristics for the problem under consideration: the improved fruit fly optimization (IFFO), the discrete artificial bee colony optimization (DABC), and the improved migrating birds optimization (IMBO).

4.1. IFFO algorithm

FFO is a new interactive evolutionary method proposed by Pan in 2012 [30]. This section presents an IFFO for the SDST/HFS problem with a makespan criterion. In Pan [30], biological terminologies are used to describe the FFO including a swarm, an osphresis foraging stage and a vision foraging stage.

These terminologies are respectively translated into a solution, an exploration strategy and an exploitation strategy in this paper to facilitate the readers' understanding. Unlike the conventional FFO that has a single solution trying to find the optimal solution, IFFO evolves with a number of solutions. The neighborhoods of these solutions are first explored independently. Then, a path relinking based cooperative search is utilized to emphasize the cooperation among the solutions. We first explain the heuristic and solution initialization, exploration and exploitation strategy, cooperative search, and diversity control strategy and then give the computational procedure of the presented IFFO algorithm.

4.1.1. Heuristics and initialization

The shortest processing time (SPT) rule has been commonly used in the scheduling literature. SPT gives higher priorities to jobs with shorter processing times, but it cannot be directly applied to the problem under consideration because SDSTs are involved. We consider the SDST/HFS problem in the first stage. If job $j \in J$ is assigned to a machine that has not processed any job before, the job can be finished at time $\psi_{1,j} = s_{1,j,j} + p_{1,j}$. It is reasonable to give higher priority to jobs with a smaller $\psi_{1,j}$ when we select the first job for a machine. The SDST/HFS has multiple stages. We consider multiple stages together to determine the first job for a machine. Let $\psi_j = \sum_{k=1}^{k^*} \psi_{k,j}$, where $\psi_{k,j} = s_{k,j,j} + p_{k,j}$. An SPT-based heuristic is presented as follows.

- Step 1. Sort jobs according to increasing $\psi_j = \sum_{k=1}^{k^*} (s_{k,j,j} + p_{k,j})$ to yield a sequence $\pi = (\pi_1, \pi_2, \dots, \pi_n)$.
- Step 2. Split $\pi = (\pi_1, \pi_2, \dots, \pi_n)$ into two sub-sequences, $\pi^1 = (\pi_1, \pi_2, \dots, \pi_{|M_1|})$ and $\pi^2 = (\pi_{|M_1|+1}, \pi_{|M_1|+2}, \dots, \pi_n)$, i.e., $\pi = \pi^1 \cup \pi^2$.
- Step 3. Randomly shuffle sub-sequence $\pi^2 = (\pi_{|M_1|+1}, \pi_{|M_1|+2}, \dots, \pi_n)$ to obtain $\pi^{2'}$. By combining π^1 and $\pi^{2'}$, we generate a solution $\pi' = \pi^1 \cup \pi^{2'}$.
- Step 4. Evaluating the generated solution π' , we obtain the whole schedule.

There are $|M_1|$ machines at stage 1, and the first $|M_1|$ jobs of $\pi = (\pi_1, \pi_2, \dots, \pi_n)$ are launched on these machines. So, in Step 2, we keep $\pi^1 = (\pi_1, \pi_2, \dots, \pi_{|M_1|})$ unchanged because we use $\psi_{1,j} = s_{1,j,j} + p_{1,j}$ to estimate the completion time for these jobs. We fix $k^* = 3$ after the primary experiments.

To initialize IFFO, we generate a number of solutions for the given problem by repeating the above heuristic NS times, where NS is the number of solutions. We keep all of the solutions different from

each other to form a population with high diversity.

4.1.2. Exploration and exploitation strategy

We perform exploration and exploitation independently for each solution in the population. In the exploration process, $\mathcal{G}$ candidate solutions are generated randomly around each old solution. To generate a candidate solution, we pick up an insertion or interchange operator with equal probability. Then, the selected operator is used to find a random neighboring solution in its neighborhood. In the exploitation phase, the candidate solution with the lowest makespan is found for each old solution. If the candidate solution is better than the old solution, it will replace the old solution and become a new solution in the next iteration.

4.1.3. Path relinking based cooperative search

Zheng *et al.* [33] introduced a cooperative search into FFO where solutions exchange information by an improved precedence operation crossover and multipoint preservative crossover. The crossovers of Zheng *et al.* [33] are not suitable for our problem. Alternatively, the path relinking presented by Glover and Laguna [47] is an effective solution-combination method that has been successfully used in the recent literature [48]. Thus, we present a path relinking-based cooperative search for IFFO.

The cooperative search sets the current solution as the starting solution. Four methods are considered to determine the target solution. The first method always sets the best solution within the population as the target solution. Every solution exchanges information with the best solution. The algorithm performs an intensive search around the best solution and reduces the path relinking search to a small area. Although a fast convergence can be achieved, the best solution locating a local optimum may lead to a premature convergence. The second method randomly chooses a solution from the population as the target solution. This method equally visits the intermediate areas within the whole population, improving the diversity of the population and favouring exploration. The third method stores all of the solutions in a ring lattice and fixes the right (or left) neighbor of the current solution as its target solution. The method further degrades the influence of the best solution on the whole population, as each solution only exchanges information with its right (or left) neighbor and has no chance to exchange information with the other solutions. This slows down the convergence of IFFO, so the algorithm has a greater opportunity to avoid local optima. The last method is based on an agent lattice, which is one key component of the knowledge-based evolution algorithm and provides a physical environment for agents [33]. Each solution is fixed on a lattice point and has four neighbors. The best neighbor of the current solution is selected as its target solution. This method can be regarded as a compromise among the above methods. We determine an appropriate method by calibration in Section 5.

After the target solution is chosen, we need to determine a path from the starting solution to the target solution. For the job permutation based representation, we perform a series of interchange moves

to transform the starting solution into the target solution. Each time we carry out an interchange move, an intermediate solution is obtained. As the number of moves increases, the attributes of the target solution are progressively introduced. The obtained intermediate solution looks more like the target solution than the starting solution. There are always multiple paths from the starting solution to the target solution. We randomly select a path to obtain the intermediate solutions. Let us consider an example with the starting solution $\pi'' = (1, 2, 3, 4, 5, 6)$ and the target solution $\pi = (2, 4, 3, 5, 6, 1)$. An interchange from the candidate set $\{(1, 2), (2, 4), (4, 5), (5, 6), (6, 1)\}$ can make solution π'' more like the target solution π . Suppose that move $(2, 4)$ is chosen by a random selection, exchanging the positions of job 2 and job 4 in the permutation. An intermediate solution $\pi' = (1, 4, 3, 2, 5, 6)$ is yielded. Then, we randomly perform a move, such as $(2, 5)$, from the candidate move set $\{(1, 2), (2, 5), (5, 6), (6, 1)\}$ to π' and obtain a new solution $(1, 4, 3, 5, 2, 6)$. Next, we let $\pi' = (1, 4, 3, 5, 2, 6)$ and repeat the above procedure. Finally, we reach the target solution $\pi = (2, 4, 3, 5, 6, 1)$ and obtain a list of intermediate solutions $\Gamma = \{(1, 4, 3, 2, 5, 6), (1, 4, 3, 5, 2, 6), (2, 4, 3, 5, 1, 6)\}$.

During the path relinking procedure, we obtain a list of the intermediate solutions Γ . To avoid a cycling search around a local optimum, we exclude the first and last intermediate solutions from the set Γ , unless they improve their neighboring solutions. Naturally, the algorithm can be further enhanced by applying a local search to the intermediate solutions to visit the locally optimal areas along the path [47]. It is not productive to apply a local search to each intermediate solution, because many of these solutions differ only by interchanging two elements, and the local searches would lead to the same local optimum. The application of a local search procedure to all of the intermediate solutions would also be very time consuming [48]. The best solution often implies its most promising area, so we only perform the local search to the best solution from set Γ . For the local search, we employ the exploration and exploitation strategy in Section 4.1.2 for simplicity. The resulting solution after the local search will replace the starting solution of the path and become a new solution in the next generation if the resulting solution is better than the starting solution.

To develop the path-relinking operators, we should keep the starting solution π'' and the target solution π different, *i.e.*, $\pi'' \neq \pi$. There also exists a case where transforming π'' to π needs only a few moves, and the intermediate set is empty. In this case, we perform two random interchanges for the starting solution π'' to generate a new solution. The complete computational procedure of the

presented path relinking the cooperative search algorithm is outlined in Fig. 16.

[Insert Fig. 16 here]

4.1.4 Diversity control strategy

As the population evolves after a number of generations, the solutions in the population become very similar. To address this problem, we present a diversity control scheme. At each generation, we determine the best makespan found so far. If the best makespan found so far has not been improved upon, we increment a counter. When the counter becomes larger than a threshold value, $\gamma_{\max}$, we abandon all solutions and introduce new solutions to the population. However, if all new solutions are generated randomly in the solution space, the search efficacy will be decreased, because the useful information collected by the abandoned solutions is not well-utilized. The abandoned solutions survived many generations, so they should carry good and promising information. In the experiments, we find that there are many similar blocks of jobs within the solutions, as was also observed by Ruiz and Maroto [10] and Jarboui *et al.* [49]. These blocks may be located at the same positions or different positions. If these blocks are broken after reproduction, an offspring with a worse makespan is often generated. Therefore, we propose to construct a new solution, taking advantage of both the abandoned solution and the information of the building blocks of jobs.

Let $[\delta_{j',j}]_{n \times n}$ be a job block matrix containing n rows and n columns, where element $\delta_{j',j}$ represents the number of times that job j appears immediately after job j' in all of the solutions. We yield a subsequence from $\pi = (\pi_1, \pi_2, \dots, \pi_n)$ and try to keep the building blocks of the jobs. For each job π_k of π , we calculate the remaining probability μ_k . Then, a random number is generated between 0 and 1. If it is less than μ_k , the job remains unchanged in the permutation; otherwise, it is removed. The remaining probability is given by

$$\mu_k = \begin{cases} 0.1 + \frac{\delta_{\pi_1, \pi_2} - \delta_{\min}}{\delta_{\max} - \delta_{\min}} \times 0.8 & \text{if } k = 1 \\ 0.1 + \frac{(\delta_{\pi_{k-1}, \pi_k} + \delta_{\pi_k, \pi_{k+1}}) / 2 - \delta_{\min}}{\delta_{\max} - \delta_{\min}} \times 0.8 & \text{if } 1 < k < n \\ 0.1 + \frac{\delta_{\pi_{n-1}, \pi_n} - \delta_{\min}}{\delta_{\max} - \delta_{\min}} \times 0.8 & \text{if } k = n \end{cases} \quad (6)$$

where $\delta_{\max} = \max_{\substack{k=1,2,\dots,n \\ k'=1,2,\dots,n}} (\delta_{\pi_k, \pi_{k'}})$, $\delta_{\min} = \min_{\substack{k=1,2,\dots,n \\ k'=1,2,\dots,n}} (\delta_{\pi_k, \pi_{k'}})$.

In Eqn (6), the remaining probability μ_k is computed with regard to $\delta_{\pi_{k-1}, \pi_k}$ and $\delta_{\pi_k, \pi_{k+1}}$ for

$1 < k < n$, as job π_k is included in blocks (π_{k-1}, π_k) and (π_k, π_{k+1}) . For the remaining probability μ_1 or μ_n , only block (π_1, π_2) or (π_{n-1}, π_n) needs to be considered. In each sub-equation, a factor 0.8 is employed to limit the remaining probability within the range of $[0.1, 0.9]$. In this way, the block with the greatest occurrence has a chance to be broken, whereas the block with the smallest occurrence has a chance to be kept, thereby introducing diversity. After obtaining the subsequence from an abandoned solution, we construct a new sequence by randomly inserting the missing jobs into randomly selected slots in the subsequence.

In the evolution process, every time the best makespan found so far does not change for more than $\gamma_{\max}$ generations, our diversity control procedure is invoked. The 50% best solutions are replaced with the new solutions generated by the above building block based method, and the remaining 50% are replaced by new randomly generated schedules. Let $\Pi = \{\Pi_1, \Pi_2, \dots, \Pi_{NS}\}$ be the current population. The diversity control scheme is described in Fig. 17.

[Insert Fig. 17 here]

4.1.5 Procedure of IFFO for the SDST/HFS problem

The complete computational procedure of IFFO is outlined in Fig. 18.

[Insert Fig. 18 here]

4.2. IMBO algorithm

The migrating birds optimisation (MBO) presented by Duman *et al.* [31] is another population-based metaheuristic. It was inspired by the V flight formation of migrating birds. In our previous study [33], we present an improved MBO (IMBO) algorithm for the hybrid flowshop scheduling problem with a total flowtime criterion. We introduced some advanced and effective technologies including a diversified initialisation approach, a mixed neighborhood structure, and a leaping mechanism to improve the MBO. Our previous experiments demonstrated that the IMBO was a new state-of-the-art algorithm for the hybrid flowshop scheduling problem with a total flowtime criterion. IMBO makes few assumptions about the optimisation problem being solved, so we can adapt IMBO to the SDST/HFS problem under consideration by only changing the original objective evaluation function to our objective evaluation function. We will calibrate IMBO in Section 5. For details on IMBO, the readers can refer to Pan and Dong [34].

4.3. DABC algorithm

The artificial bee colony (ABC) algorithm was inspired by the specific aggregating behaviors of honeybee swarms. It was proposed by Karaboga [32] to optimise multi-variable and multi-modal continuous functions. Our previous study [35] proposed a discrete artificial bee colony (DABC) algorithm for the hybrid flowshop problem with a makespan criterion. In the DABC, we developed a hybrid presentation that combines forward decoding and backward decoding methods. We generated an initial population with a high level of quality and diversity based on problem-specific heuristics. We introduced a new control parameter to conduct the search of the employed bees and onlooker bees with the intention of balancing the global exploration and local exploitation. We presented an enhanced strategy for the scout bee phase to prevent the algorithm from searching in poor regions of the solution space. We developed a problem-specific local refinement procedure to search for regions of the solution space that are not explored by the honey bees. Our extensive computational experiments showed that DABC was very effective in solving the HFS problem with the makespan criterion. In this paper, we adapt DABC to the considered SDST/HFS problem with the following changes: the original hybrid representation is replaced with the solution representation in Section 3.1; the original initialization method is replaced with the initialization method presented in Section 4.1.1; the problem-specific local refinement procedure is omitted; and the original objective evaluation function is replaced with our objective evaluation function. That is, DABC starts from an initial population generated by the method in Section 4.1.1. Then, the employed bee stage, onlooker bee stage and scout bee stage, proposed in Pan *et al.* [35], are consecutively performed until a termination criterion is reached. Meanwhile, the balance strategy of exploration and exploitation of Pan *et al.* [35] is utilized to achieve a high-performing algorithm. The readers can refer to Pan *et al.* [35] for details on DABC.

5. Experimental Calibration

We calibrate all of the presented nine algorithms by means of a design of experiments (DOE) approach and analysis of variance (ANOVA). We only detail the calibration of IFFO for reasons of space, but all experimental results are available upon request from the authors.

IFFO has three control parameters: NS , $\mathcal{G}$, and $\gamma_{\max}$. After preliminary experiments, we test NS at three levels, 64, 81, and 100, $\mathcal{G}$ at four levels, $\frac{n}{2}$, $\frac{n}{3}$, $\frac{n}{4}$, and $\frac{n}{5}$, and $\gamma_{\max}$ at five levels, n , $2n$, $3n$, $4n$ and $+\infty$, where $+\infty$ denotes IFFO without the diversification strategy. We also test the selection method of the target solution in the path relinking procedure. As mentioned in Section 4.1.3, we consider four methods that select the best solution, a random solution, the right neighbor, and the best neighbor based on an agent lattice. All of the possible combinations of the above factors result in a total of $4 \times 3 \times 5 \times 4 = 240$ different configurations for IFFO.

We consider a full factorial design, and for each of the 240 configurations, we solve a set of 60 instances. For the instances, we consider five different configurations for the number of jobs

$n \in \{20, 40, 60, 80, 100\}$ and three for the number of stages $m \in \{5, 8, 10\}$, for a total of 15 different configurations. For each configuration, we generate a problem where the processing times are uniformly distributed in the range of $[1, 99]$, *i.e.*, $p_{k,j} \in U[1, 99]$ and the number of parallel machines at each stage is uniformly distributed between 1 and 5, *i.e.*, $|M_k| \in U[1, 5]$. Four instances with different setup times are generated for each configuration, where the setup times are drawn from the uniform distributions $U[1, 25]$, $U[1, 49]$, $U[1, 99]$, and $U[1, 124]$. Thus, on average, the setup times correspond to 25%, 50%, 100%, and 125% of the average processing times.

We code the IFFO in Microsoft Visual Studio 2013. All 240 configurations are run on the same Intel (R) Core (TM) i7-2600 CPU @ 3.40 GHz with 8.00 GB RAM in the Windows 7 Operation System. The algorithms stop if the termination criterion of the maximum elapsed CPU time of $50 \cdot n \cdot m$ milliseconds is met. The relative percentage increase (RPI) is calculated as a response variable as

$$RPI(F) = (F - F^*) / F^* \times 100 \quad (7)$$

where F is the solution generated by a given algorithm configuration, and F^* is the best solution found in the experiment.

Following Ruiz and Stützle [22, 23], Pan and Ruiz [26, 27], and many others, we carry out the experiments by ANOVA and analyse the experimental results using the F -ratio for those factors and interactions whose p -value is close to zero. The means plots of those factors and interactions are shown in Fig. 19 with Tukey's Honestly Significant Difference (HSD) confidence intervals at the 95% confidence level.

The most influential factor turns out to be the selection method of the target solution in the path relinking procedure. It can be observed from Fig. 19 a) that there is a clear statistically significant difference among the four different methods (the non-overlapping intervals suggest a statistically significant difference at the 95% confidence level). The first method that always sets the best solution as the target solution yields the worst IFFO, whereas the second method that randomly selects a solution and the third method that chooses the right neighbor produce the two best IFFO algorithms, with the latter being significantly better than the former.

The second most influential factor is the threshold value $\gamma_{\max}$. It can be seen from the means plot in Fig. 19 b) that the performance of IFFO seriously degrades if the diversity control strategy is not adopted, *i.e.*, $\gamma_{\max} = +\infty$, demonstrating the effectiveness of using the diversification strategy in the evolution process. For $\gamma_{\max}$ from n to $4n$, IFFO produces the best results with $\gamma_{\max} = n$.

We proceed with the analysis of the influence of the number of solutions, NS . We can observe from

Fig. 19 c) that a smaller value $NS = 64$ leads to a significantly better IFFO than the other values. However, from Fig. 19 d), the size of the solution has no significant effect on IFFO, suggesting that the IFFO is relatively robust when $\mathcal{G}$ is in the range of $[n/5, n/2]$. Although $\mathcal{G} = n/3$ generates a lower average RPI than the other values, the interactions of NS and $\mathcal{G}$ show that $\mathcal{G} = n/2$ works much better when $NS = 64$. So, we set $\mathcal{G} = n/2$ for this parameter.

Finally, we set the parameters of IFFO as $\gamma_{\max} = n$, $\mathcal{G} = n/2$, and $NS = 64$ and set the right neighbor of the current solution as its target solution in the path relinking procedure.

[Insert Fig. 19 here]

A similar method is used to calibrate the remaining eight algorithms. The parameters and operators of the presented algorithms are summarized in Table 3.

[Insert Table 3 here]

6. Computational evaluation

We test the performance of the presented nine algorithms by extensive numerical comparisons. We compare them with several well-performing meta-heuristics found in the literature: the immune algorithm (IA) of Zandieh *et al.* [8], the GA of Ruiz and Maroto [10], the ILSN of Naderi *et al.* [11], the ISA of Naderi and Zandieh [12], the artificial immune system (AIS) of Engin and Doyen [50], and the immunoglobulin-based artificial immune system (IAIS) of Chung and Liao [51]. IA, GA, ILSN, and ISA are presented for SDST/HFS problems, whereas AIS and IAIS are for general HFS problems. We also consider the FFO algorithm (namely, nFFO) presented for the semiconductor final testing scheduling problem in Zheng *et al.* [32], the multi-restart iterated local search of Dong *et al.* (ILSD for short, where the capital letter D is taken from authors' name) [24] for the permutation flowshop problem, and the variable iterated greedy (vIG) for the traveling salesman problem with time windows by Karaulut and Tasgetiren [52]. We change the way that the makespan is calculated by the one shown in Section 4.1.1 for the above adapted algorithms. As done in Section 5, we recalibrate the algorithmic parameters and operators by ANOVA. For nFFO, because the original precedence operation crossover and multipoint preservative crossover cannot be directly used for the job permutation, we test the commonly used crossover operators including one-point crossover, two-point crossover, and the partial map crossover.

To evaluate these algorithms, we randomly generate a set of 300 instances with $n \in \{20, 40, 60, 80, 100\}$ and $m \in \{5, 8, 10\}$, for a total of 15 combinations of n and m . For each combination, we generate 5 instances with $p_{k,j} \in U[1, 99]$ and $|M_k| \in U[1, 5]$. Therefore, we have

$15 \times 5 = 75$ instances in total. For each instances, we consider four uniform distributions for setup times: $s_{k,i,j} \in U[1,25]$, $s_{k,i,j} \in U[1,49]$, $s_{k,i,j} \in U[1,99]$ and $s_{k,i,j} \in U[1,124]$. Note that the above instances are quite different from those in Section 5 since we employ different random number seeds when generate them. We carry out four different evaluations, each with one setup time distribution. All of the algorithms adopt the same maximum elapsed CPU time limit of $t = \rho \cdot m \cdot n$ milliseconds as a termination criterion, where ρ has been tested at 12 values, 25, 50, 75, 100, 125, 150, 175, 200, 225, 250, 275, and 300, to provide an overall picture of the performance of the competing algorithms. All of the algorithms have been coded in Microsoft Visual Studio 2013, and the experiments are carried out on the same PC as described above. We carry out 5 independent runs for each algorithm for each of the 75 instances and adopt the minimal makespan found at $\rho = 300$ as F^* to compute the RPI values. The average computed results and their ranks (in brackets) are reported in Tables 4-7. In the tables, each cell is the average RPI over 375 results (combinations of instances and replications, *i.e.*, $75 \times 5 = 375$).

Table 4 shows the average RPI values and their ranks generated by the 18 competing algorithms at the 12 different elapsed CPU times for the instances with 25% setup times. It can be observed that all of the proposed algorithms perform much better than the existing population-based metaheuristics including GA, nFOA, AIS, IAIS, and IA in terms of the RPI values, regardless of the CPU time involved. The proposed DABC, IG_{MRLS}, ILS_{MRLS}, and ILS_{HLS}, from best to worst, rank as the top four among the 18 competing algorithms. IG_{HLS} ranks fifth for $\rho \leq 225$, but it is surpassed by ILSD when $\rho > 225$. IG_{VNS} and ILS_{VNS} are the worst two of the proposed algorithms and are surpassed by ILSN, vIG, and ILSD.

[Insert Table 4 here]

We further carry out a multifactor ANOVA to determine whether the observed differences from Table 4 are indeed significantly different. The types of algorithms are considered as factors. We investigate the performance of the competing algorithms over the 12 different elapsed CPU times. IA is ruled out because its results are clearly worse, so it is not necessary to test for significance. The ANOVA results show that different types of algorithm result in statistically significant differences in the response variable (RPI) at a 95% confidence level. Fig. 20 reports the means plots together with the 95% HSD confidence intervals of the algorithms. It can be observed that the competing algorithms can be divided into 14 homogenous groups, with no significant differences within each group. These groups, from best to worst, are {DABC}, {IG_{MRLS}}, {ILS_{MRLS}}, {ILS_{HLS}}, {IG_{HLS}}, {ILSD, IMBO}, {IFFO, vIG, ILSN}, {IG_{VNS}}, {ILS_{VNS}}, {nFOA}, {GA}, {ISA}, {IAIS}, and {AIS}.

[Insert Fig. 20 here]

The computational results for the instances with 50%, 100% and 125% setup times are reported in Tables 5-7. The means plots with 95% HSD confidence intervals for the instances with 50%, 100% and

125% setup times are reported in Figs. 21-23. As we can see, DABC, IG_{MRLS} , ILS_{MRLS} , and ILS_{HLS} are the top four performing algorithms with different setup times involved. DABC algorithm significantly outperforms the other competing algorithms for solving the SDST/HFS problem with the makespan criterion.

[Insert Table 5 here]

[Insert Table 6 here]

[Insert Table 7 here]

[Insert Fig. 21 here]

[Insert Fig. 22 here]

[Insert Fig. 23 here]

Vallada *et al.* (2003) [39] published a database for the SDST/HFS problems but with unrelated parallel machines during production stages. The database is an adaption of the dataset used in Taillard [51]. Four types of sequence-dependent setup time are considered, corresponding to 10%, 50%, 100%, and 125% of the processing times (termed SSD10, SSD50, SSD100 and SSD125, respectively). Two cases of the numerical distribution of the machines per stage are P13 (*i.e.*, randomly distributed between one and three machines) and P3 (*i.e.*, a constant number of three machines). Gomez-Gasquet *et al.* (2012) [9] considered two load levels in the workshop (20 and 50 jobs) for the database, and they obtained a subset of the database including 240 instances in total. After computational experiments, Gómez-Gasquet *et al.* (2012) [9] reported the best known makespan values for these 240 instances.

Following Gómez-Gasquet *et al.* (2012) [9], we solve the 240 benchmark instances using the presented DABC algorithm. We employ the same control parameters for the algorithm as those in Section 5.2 and run the algorithm with the maximum elapsed CPU time limit of $300mn$ milliseconds. The computational results reveal that our DABC algorithm improves 126 best known makespan values among the 240 benchmark instances.

[Insert Table 8 here]

[Insert Table 9 here]

7. Conclusions

The hybrid flowshop scheduling problem has important applications in the chemical, textile, glass, and other industries. In this paper, we present a total of nine algorithms to minimize the makespan for the hybrid flowshop problem with sequence-dependent setup times: iterated local search with hybrid local search, iterated local search with variable neighborhood search, iterated local search with mixed

referenced local search, iterated greedy with hybrid local search, iterated greedy with variable neighborhood search, iterated greedy with mixed referenced local search, improved fruit fly optimization, improved migrating birds optimization, and discrete artificial bee colony optimization. We calibrate their parameters and operators by means of a design of experiments approach and carry out extensive comparative evaluations of the proposed algorithms against the most recent techniques for the considered problem in the literature. The experimental results and comparisons demonstrated the effectiveness of the proposed algorithms for the SDST/HFS problem with the makespan criterion.

The proposed algorithms fall into the category of stochastic local searches (SLSs), which make use of randomized choices in generating or selecting candidate solutions in the search process [54]. It could be interesting to attempt to combine exact algorithms from the mixed integer programming paradigm, such as branch-and-bound and branch-and-cut, to improve their performance [55, 56]. It could be also interesting to apply the proposed algorithm to other optimization problems including hybrid flowshop scheduling problem with due date window [57], assignment optimization in a multi-tier shuttle warehousing system [58], optimization of the dressing parameters [59], etc.

Acknowledgements

This research is partially supported by the National Natural Science Fund for Distinguished Young Scholars of China 61525304, the National Science Foundation of China 51575212 and 61603169.

References

- [1] M. Pinedo, *Scheduling: Theory, Algorithms and Systems* (Second ed.), Prentice-Hall, Englewood Cliffs, NJ 2002.
- [2] C. Rajendran, D. Chaudhuri, Scheduling in n -jobs m stage flow shop with parallel processors to minimize makespan, *International Journal of Production Economics*, 27 (1992) 137-143.
- [3] R. L. Graham, E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, Optimization and approximation in deterministic sequencing and scheduling: a survey, *Annals of Discrete Mathematics*, 5 (1979) 287-362.
- [4] A. Vignier, J. C. Billaut, C. Proust, Les problèmes d'ordonnancement de type flow-shop hybride: état de l'art. *RAIRO Recherche opérationnelle*, 33 (1999) 117-183, in French.
- [5] J. N. D. Gupta, Two-stage hybrid flowshop scheduling problem, *Journal of the operational Research Society*, 39 (1988) 359-364.
- [6] M. E. Kurz, R. G. Askin, Comparing scheduling rules for flexible flow lines, *International Journal of Production Economics*, 85 (2003) 371-388.
- [7] M. E. Kurz, R. G. Askin, Scheduling flexible flow lines with sequence-dependent setup times, *European Journal of Operational Research*, 159 (2004) 66-82.
- [8] M. Zandieh, S. M. T. F. Ghomi, S. M. M. Hussein, An immune algorithm approach to hybrid flow shops scheduling with sequence-dependent setup times, *Applied Mathematics and Computation*, 180 (2006) 111-127.
- [9] P. Gómez-Gasquet, C. Andrés, F. C. Lario, An agent-based genetic algorithm for hybrid flowshops with sequence dependent setup times to minimise makespan, *Expert Systems with Applications*, 39 (2012) 8095-8107.
- [10] R. Ruiz, C. Maroto, A genetic algorithm for hybrid flowshops with sequence dependent setup times and

- machine eligibility, *European Journal of Operational Research*, 169 (2006) 781-800.
- [11] B. Naderi, R. Ruiz, M. Zandieh, Algorithms for a realistic variant of flowshop scheduling, *Computers & Operations Research*, 37 (2010) 236-246.
 - [12] B. Naderi, M. Zandieh, A. K. G. Balagh, V. Roshanaei, An improved simulated annealing for hybrid flowshops with sequence-dependent setup and transportation times to minimize total completion and total tardiness, *Expert Systems with Applications*, 36 (2009) 9625-9633.
 - [13] R. Ruiz, F. Sivrikaya Serifoglu, T. Urlings, Modeling realistic hybrid flexible flowshop scheduling problems, *Computers & Operational Research*, 35 (2008) 1143-1159.
 - [14] J. Jungwattanakit, M. Reodecha, P. Chaovalitwongse, F. Werner. An Evaluation of Sequencing Heuristics for Flexible Flowshop Scheduling Problems with Unrelated Parallel Machines and Dual Criteria, in the Proceeding of the 12th IFAC/IFIP/IFORS/IEEE Symposium on Information Control Problems in Manufacturing (INCOM 2006), MAY 17-JUL 19, 2006 , St Etienne, FRANCE.
 - [15] J. Jungwattanakit, M. Reodecha, P. Chaovalitwongse, F. Werner, Algorithms for flexible flow shop problems with unrelated parallel machines, setup times, and dual criteria, *International Journal of Advanced Manufacturing Technology*, 37 (2008) 354-370.
 - [16] E. Ebrahimi, S. M. T. Fatemi Ghomi, B. Karimi, Hybrid flow shop scheduling with sequence dependent family setup time and uncertain due dates, *Applied Mathematical Modelling*, 38 (2014) 2409-2504.
 - [17] N. Srinivas, K. Deb, Multi-objective function optimization using non-dominated sorting genetic algorithms, *Evolutionary Computation*, 2 (1995) 221-248.
 - [18] T. Murata, H. Ishibuchi, H. Tanaka, Multi-objective genetic algorithm and its application to flow shop scheduling, *Computers and Industrial Engineering*, 30 (1996) 957-968.
 - [19] I. Ribas, R. Leisten, J. M. Framinan, Review and classification of hybrid flow shop scheduling problems from a production system and a solutions procedure perspective, *Computers & Operations Research*, 37 (2010) 1439-1454.
 - [20] R. Ruiz, J. A. Vazquez Rodriguez, The hybrid flow shop scheduling problem, *European Journal of Operational Research*, 205 (2010) 1-18.
 - [21] H. R. Lourenço, O. C. Martin, T. Stützle, Iterated local search, Chapter 11 in F. W. Glover, G. A. Kochenberger, (Eds) *Handbook of Metaheuristics*, International Series in Operations Research & Management Science, Kluwer Academic Publishers, Norwell, MA, 57 (2003) 321-353.
 - [22] R. Ruiz, T. Stützle, A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. *European Journal of Operational Research*, 177(2007) 2033-2049.
 - [23] R. Ruiz, T. Stützle, An iterated greedy heuristic for the sequence dependent setup times flowshop problem with makespan and weighted tardiness objectives. *European Journal of Operational Research*, 187 (2008) 1143-1159.
 - [24] X. Dong, N. Maciek, P. Chen, Y. Lin, Self-adaptive perturbation and multi-neighborhood search for iterated local search on the permutation flow shop problem, *Computers & Operations Research*, 87 (2015) 176-185.
 - [25] X. Dong, N. Maciek, P. Chen, Y. Lin, A multi-restart iterated local search algorithm for the permutation flow shop problem minimizing total flow time. *Computers & Operations Research*, 2013, 40, 627-632.
 - [26] Q. K. Pan, R. Ruiz, An effective iterated greedy algorithm for the mixed no-idle permutation flowshop scheduling problem, *OMEGA-International Journal of Management Science*, 44 (2014) 41-50.
 - [27] Q. K. Pan, R. Ruiz, Local search methods for the flowshop scheduling problem with flowtime minimization, *European Journal of Operational Research*, 222 (2012) 31-41.
 - [28] M. F. Tasgetiren, Q. K. Pan, P. N. Suganthan, O. Buyukdagli, A Variable Iterated Greedy Algorithm with Differential Evolution for the No-Idle Permutation Flowshop Scheduling Problem, *Computers & Operations*

Research, 40 (2013) 1729-1743.

- [29] T. Qin, B. Peng, U. Benlic, T.C.E. Cheng, Y. Wang, Z. Lü, Iterated local search based on multi-type perturbation for single-machine earliness/tardiness scheduling, *Computers & Operations Research*, 61 (2015) 81-88.
- [30] W. T. Pan, A new fruit fly optimization algorithm: taking the financial distress model, *Knowledge-based Systems*, 26 (2012) 69-74.
- [31] E. Duman, M. Uysal M, A. F. Alkaya, Migrating birds optimization: A new metaheuristic approach and its performance on quadratic assignment problem, *Information Sciences*, 217 (2012) 65-77
- [32] D. Karaboga. An idea based on honey bee swarm for numerical optimization. Technical report TR06, Erciyes University, Engineering Faculty, Computer Engineering Department, 2005
- [33] X. L. Zheng, L. Wang, S.Y. Wang, A novel fruit fly optimization algorithm for the semiconductor final testing scheduling problem, *Knowledge-based Systems*, 57 (2014) 95-103.
- [34] Q. K. Pan, Y. Dong, An improved migrating birds optimisation for a hybrid flowshop scheduling with total flowtime minimization. *Information Sciences*, 277 (2014) 67-77.
- [35] Q. K. Pan, L. Wang, J. Q. Li, A novel discrete artificial bee colony algorithm for the hybrid flowshop scheduling problem with makespan minimization, *OMEGA-International Journal of Management Science*, 45 (2014) 42-56.
- [36] M. F. Tasgetiren, Q. K. Pan, P. N. Suganthan, A. H. L. Chen. A discrete artificial bee colony algorithm for the total flowtime minimization in permutation flow shops. *Information Sciences*, 181 (2011) 3459-3475.
- [37] X. Wang, X. Xie, T. C. E. Cheng, A modified artificial bee colony algorithm for order acceptance in two-machine flow shops. *International Journal of Production Economics*, 141 (2013) 12-23.
- [38] Q. K. Pan, An effective co-evolutionary artificial bee colony algorithm for steelmaking-continuous casting scheduling, *European Journal of Operational Research* 250 (2016) 702-714.
- [39] E. Vallada, R. Ruiz, C. Maroto, Synthetic and Real Benchmarks for Complex Flow-shops Problems. Technical Report, Grupo de Investigación Operativa (GIO), Universitat Politècnica de València.
- [40] P. Hansen, N. Mladenovic, Variable neighborhood search: Principles and applications, *European Journal of Operational Research* 130 (2001) 449-467.
- [41] M. Tasgetiren, Y. C. Liang, M. Sevkli, G. Gencyilmaz, A particle swarm optimization algorithm for makespan and total flowtime minimization in the permutation flowshop sequencing problem. *European Journal of Operational Research*, 177 (2007)1930-1947.
- [42] B. Naderi, R. Ruiz, The distributed permutation flowshop scheduling problem. *Computers & Operations Research*, 37 (2010) 754-768.
- [43] M. F. Tasgetiren, Q. K. Pan, P. N. Suganthan, A. H. L. Chen, A discrete artificial bee colony algorithm for the total flowtime minimization in permutation flow shops. *Information Sciences*, 181 (2011) 3459-3475.
- [44] Q. K. Pan, M. F. Tasgetiren, Y. C. Liang, A discrete differential evolution algorithm for the permutation flowshop scheduling problem. *Computers & Industrial Engineering*, 55 (2008) 795-816.
- [45] M. Nawaz, E. E. J. Enscore, I. Ham, A heuristic algorithm for the m -machine, n -job flow shop sequencing problem. *OMEGA* 11 (1983) 91-95.
- [46] T. Stützle, Applying iterated local search to the permutation flow shop problem. Technical Report AIDA-98-04, FG Informatik, FB Informatik, TU Darmstadt, 1998.
- [47] F. Glover, M. Laguna. Tabu search. In: C. R. Reeves (Ed.), *Modern heuristic techniques for combinatorial optimization problems*, Blackwell, Oxford, pp. 70-150.
- [48] B. Peng, Z. Lu, T. C. E. Cheng, A tabu search/path relinking algorithm to solve the job shop scheduling problem, *Computers & Operations Research*, 53 (2015) 154-164.

- [49] B. Jarboui, M. Eddaly, P. Siarry, An estimation of distribution algorithm for minimizing the total flowtime in permutation flowshop scheduling problems, *Computers & Operations Research*, 36 (2009) 2638-2646.
- [50] O. Engin, A. Doyen, A new approach to solve hybrid flow shop scheduling problems by artificial immune system, *Future Generation Computer Systems*, 20 (2004) 1083-1095.
- [51] T. P. Chung, C. J. Liao, An immunoglobulin-based artificial immune system for solving the hybrid flow shop problem, *Applied Soft Computing*, 13 (2013) 3726-3736.
- [52] K. Karabulut, M. F. Tasgetiren. A variable iterated greedy algorithm for the traveling salesman problem with time windows. *Information Science*, 279 (2014) 383-395.
- [53] E. Taillard, Benchmarks for basic scheduling problems. *European Journal of Operational Research*, 64 (1993) 278-285.
- [54] H. H. Hoos, T. Stutzle. *Stochastic local search foundation and application*, 2005, Elsevier Inc.
- [55] J. Fu, J. Faust, B. Chachuat, A. Mitsos, Local optimization of dynamic programs with guaranteed satisfaction of path constraints, *Automatica*, 62 (2015) 184-192.
- [56] K. Sorensen. Metaheuristics—the metaphor exposed. *International Transaction in Operational Research*, 22 (2015) 3-18.
- [57] Q.-K. Pan, R. Ruiz, A.-F. Pedro. Iterated search methods for earliness and tardiness minimization in hybrid flowshops with due windows. *Computers & Operations Research*, 80(2017)50-60.
- [58] Y. Wang, S. Mou, Y. Wu. Assignment Optimization in a Multi-tier Shuttle Warehousing System, *Chinese Journal Of Mechanical Engineering*, 29(2016) 63-73.
- [59] A. Irina. Optimization of the Dressing Parameters in Cylindrical Grinding Based on a Generalized Utility Function, *Chinese Journal Of Mechanical Engineering*, 29(2016): 63-73.

Table 1. Processing times p_{kj} of jobs on stages 1 and 2

	J ₁	J ₂	J ₃	J ₄
Stage 1	4	5	4	3
Stage 2	5	5	3	2

Table 2. Sequence-dependent setup times s_{kij} of jobs on stages M₁ and M₂

$J_i \backslash J_j$	Stage 1				$J_i \backslash J_j$	Stage 2			
	J ₁	J ₂	J ₃	J ₄		J ₁	J ₂	J ₃	J ₄
J ₁	2	3	2	2	J ₁	2	3	2	2
J ₂	2	2	3	4	J ₂	2	1	3	4
J ₃	4	2	2	3	J ₃	3	3	2	3
J ₄	3	3	2	2	J ₄	4	3	2	2

Table 3. Calibrated parameters and operators for the nine presented algorithms

Algorithms	Parameters and operators
IFFO	$\gamma_{\max} = n$; $\vartheta = n/2$; $NS = 64$; Set the right neighbor of the current solution as its target solution in the path relinking procedure
IMBO	Population size of 21; number of neighboring solutions to be considered of 5; number of neighboring solutions to be shared with the next solution of 2; number of tours of 10; age limit of 100
DABC	Population size of 30; number of consecutive iterations in which an individual is not improved of 100; replications of the employed bee stage of 10; number of neighbors that are generated for an abandoned individual in the scout phase of 30; Solution generation operator at insertion
ILS _{HLS}	$\lambda=1.0$; $\beta=0.5$; $loop_{HLS}=n^2$; $\omega=2$; $\varpi=20$
ILS _{VNS}	$\lambda=1.0$; $loop_{VNS}=n^2$; $\omega=2$; $\varpi=10$
ILS _{MRLS}	$\lambda=1.0$; π^s =a randomly generated solution; $\alpha=4$; $\omega=2$; $\varpi=10$
IG _{HLS}	$\lambda=1.5$; $\beta=0.5$; $loop_{HLS}=0.2n^2$; $d=3$; $Enh = false$
IG _{VNS}	$\lambda=1.0$; $loop_{VNS}=n^2/2$; $d=4$; $Enh = false$
IG _{MRLS}	$\lambda=1.0$; π^s =currnt incumbent sequence; $\alpha=2$; $d=3$; $Enh = false$

Table 4. Average RPI values and their ranks (in brackets) for the instances with 25% setup times

ρ	ILS _{MRLS}	ILS _{HLS}	ILS _{VNS}	IG _{MRLS}	IG _{HLS}	IG _{VNS}	IFFO	MBO	ABC	ILSN	VIG	ILSD	GA	ISA	nFOA	AIS	IAIS	IA
25	2.42(4)	2.42(3)	3.91(13)	2.27(2)	2.49(5)	3.10(10)	3.14(11)	2.82(7)	2.07(1)	2.74(6)	2.94(9)	2.89(8)	3.96(14)	4.99(15)	3.84(12)	6.02(16)	6.06(17)	16.84(18)
50	1.98(3)	2.01(4)	3.57(13)	1.86(2)	2.11(5)	2.64(11)	2.62(10)	2.31(6)	1.68(1)	2.38(8)	2.46(9)	2.38(7)	3.75(14)	4.76(15)	3.48(12)	5.61(17)	5.47(16)	16.84(18)
75	1.74(3)	1.81(4)	3.39(13)	1.63(2)	1.90(5)	2.42(11)	2.35(10)	2.08(6)	1.49(1)	2.19(8)	2.24(9)	2.10(7)	3.62(14)	4.62(15)	3.34(12)	5.44(17)	5.17(16)	16.84(18)
100	1.58(3)	1.66(4)	3.23(12)	1.47(2)	1.77(5)	2.26(11)	2.15(10)	1.94(7)	1.34(1)	2.09(9)	2.08(8)	1.93(6)	3.55(14)	4.51(15)	3.26(13)	5.31(17)	4.98(16)	16.83(18)
125	1.47(3)	1.56(4)	3.11(12)	1.36(2)	1.67(5)	2.16(11)	2.02(9)	1.82(7)	1.24(1)	2.02(10)	1.96(8)	1.79(6)	3.48(14)	4.44(15)	3.20(13)	5.20(17)	4.86(16)	16.83(18)
150	1.39(3)	1.49(4)	3.02(12)	1.26(2)	1.60(5)	2.05(11)	1.90(9)	1.73(7)	1.16(1)	1.97(10)	1.87(8)	1.67(6)	3.41(14)	4.38(15)	3.16(13)	5.10(17)	4.77(16)	16.83(18)
175	1.32(3)	1.43(4)	2.95(12)	1.18(2)	1.53(5)	1.98(11)	1.80(8)	1.66(7)	1.11(1)	1.93(10)	1.80(9)	1.58(6)	3.35(14)	4.33(15)	3.13(13)	5.03(17)	4.69(16)	16.83(18)
200	1.27(3)	1.37(4)	2.89(12)	1.13(2)	1.48(5)	1.91(11)	1.70(8)	1.61(7)	1.05(1)	1.88(10)	1.73(9)	1.50(6)	3.32(14)	4.28(15)	3.11(13)	4.98(17)	4.60(16)	16.83(18)
225	1.21(3)	1.32(4)	2.84(12)	1.08(2)	1.44(5)	1.85(10)	1.63(8)	1.57(7)	1.01(1)	1.85(11)	1.67(9)	1.45(6)	3.27(14)	4.25(15)	3.10(13)	4.91(17)	4.54(16)	16.83(18)
250	1.15(3)	1.27(4)	2.79(12)	1.04(2)	1.40(6)	1.81(10)	1.56(8)	1.53(7)	0.97(1)	1.82(11)	1.60(9)	1.39(5)	3.23(14)	4.22(15)	3.09(13)	4.86(17)	4.48(16)	16.83(18)
275	1.11(3)	1.23(4)	2.76(12)	1.00(2)	1.36(6)	1.77(10)	1.50(8)	1.49(7)	0.94(1)	1.79(11)	1.55(9)	1.32(5)	3.18(14)	4.18(15)	3.08(13)	4.83(17)	4.44(16)	16.83(18)
300	1.08(3)	1.20(4)	2.72(12)	0.96(2)	1.33(6)	1.74(10)	1.44(7)	1.46(8)	0.92(1)	1.76(11)	1.52(9)	1.28(5)	3.16(14)	4.16(15)	3.08(13)	4.80(17)	4.39(16)	16.83(18)

Table 5. Average RPI values and their ranks (in brackets) for the instances with 50% setup times

ρ	ILS _{MRLS}	ILS _{HLS}	ILS _{VNS}	IG _{MRLS}	IG _{HLS}	IG _{VNS}	IFFO	MBO	ABC	ILSN	VIG	ILSD	GA	ISA	nFOA	AIS	IAIS	IA
25	4.12(3)	4.16(4)	6.31(12)	4.02(2)	4.34(5)	5.22(10)	5.39(11)	4.80(7)	3.54(1)	4.70(6)	5.04(9)	4.96(8)	6.62(14)	6.76(15)	6.41(13)	9.50(16)	9.86(17)	26.35(18)
50	3.33(3)	3.47(4)	5.69(12)	3.26(2)	3.67(5)	4.49(10)	4.50(11)	4.01(6)	2.89(1)	4.16(8)	4.26(9)	4.06(7)	6.22(14)	6.34(15)	5.83(13)	8.93(16)	8.96(17)	26.35(18)
75	2.92(3)	3.10(4)	5.33(12)	2.86(2)	3.28(5)	4.10(11)	4.06(10)	3.59(7)	2.55(1)	3.87(8)	3.87(9)	3.54(6)	5.96(14)	6.11(15)	5.60(13)	8.56(17)	8.48(16)	26.35(18)
100	2.68(3)	2.85(4)	5.10(12)	2.61(2)	3.00(5)	3.81(11)	3.71(10)	3.33(7)	2.34(1)	3.66(9)	3.59(8)	3.23(6)	5.80(14)	5.95(15)	5.48(13)	8.32(17)	8.15(16)	26.35(18)
125	2.48(3)	2.66(4)	4.92(12)	2.40(2)	2.83(5)	3.63(11)	3.45(9)	3.15(7)	2.20(1)	3.54(10)	3.42(8)	3.03(6)	5.69(14)	5.85(15)	5.38(13)	8.16(17)	7.95(16)	26.35(18)
150	2.33(3)	2.52(4)	4.74(12)	2.26(2)	2.70(5)	3.48(11)	3.25(8)	3.02(7)	2.08(1)	3.43(10)	3.26(9)	2.84(6)	5.58(14)	5.75(15)	5.32(13)	8.07(17)	7.80(16)	26.35(18)
175	2.21(3)	2.40(4)	4.64(12)	2.13(2)	2.58(5)	3.35(11)	3.06(8)	2.90(7)	1.97(1)	3.33(10)	3.12(9)	2.69(6)	5.51(14)	5.68(15)	5.29(13)	7.97(17)	7.64(16)	26.35(18)
200	2.09(3)	2.32(4)	4.56(12)	2.04(2)	2.44(5)	3.24(10)	2.91(8)	2.81(7)	1.87(1)	3.24(11)	3.02(9)	2.54(6)	5.44(14)	5.63(15)	5.25(13)	7.91(17)	7.53(16)	26.35(18)
225	2.02(3)	2.21(4)	4.45(12)	1.95(2)	2.36(5)	3.16(10)	2.79(8)	2.71(7)	1.80(1)	3.17(11)	2.92(9)	2.46(6)	5.38(14)	5.56(15)	5.23(13)	7.86(17)	7.42(16)	26.35(18)
250	1.94(3)	2.17(4)	4.36(12)	1.90(2)	2.28(5)	3.08(10)	2.66(8)	2.64(7)	1.75(1)	3.10(11)	2.83(9)	2.37(6)	5.34(14)	5.50(15)	5.21(13)	7.79(17)	7.34(16)	26.35(18)
275	1.84(3)	2.11(4)	4.26(12)	1.82(2)	2.20(5)	3.02(10)	2.54(7)	2.56(8)	1.69(1)	3.04(11)	2.77(9)	2.30(6)	5.32(14)	5.46(15)	5.20(13)	7.75(17)	7.29(16)	26.35(18)
300	1.77(3)	2.07(4)	4.20(12)	1.75(2)	2.14(5)	2.95(10)	2.46(7)	2.48(8)	1.64(1)	2.99(11)	2.71(9)	2.24(6)	5.29(14)	5.43(15)	5.19(13)	7.69(17)	7.23(16)	26.35(18)

Table 6. Average RPI values and their ranks (in brackets) for the instances with 100% setup times

ρ	ILS _{MRLS}	ILS _{HLS}	ILS _{VNS}	IG _{MRLS}	IG _{HLS}	IG _{VNS}	IFFO	MBO	ABC	ILSN	VIG	ILSD	GA	ISA	nFOA	AIS	IAIS	IA
25	6.93(3)	7.09(4)	10.13(13)	6.50(2)	7.26(5)	8.96(11)	8.63(10)	7.76(7)	5.97(1)	7.59(6)	7.99(9)	7.97(8)	10.88(15)	9.05(12)	10.47(14)	14.63(16)	15.39(17)	40.19(18)
50	5.74(3)	6.04(4)	9.14(13)	5.42(2)	6.13(5)	7.94(11)	7.19(10)	6.29(6)	4.95(1)	6.70(8)	6.85(9)	6.61(7)	10.30(15)	8.38(12)	9.63(14)	13.81(16)	14.11(17)	40.19(18)
75	5.19(3)	5.50(4)	8.61(13)	4.87(2)	5.58(5)	7.34(11)	6.41(10)	5.70(6)	4.41(1)	6.16(8)	6.22(9)	5.88(7)	9.92(15)	7.99(12)	9.28(14)	13.33(16)	13.41(17)	40.19(18)
100	4.75(3)	5.14(4)	8.27(13)	4.52(2)	5.17(5)	6.93(11)	5.91(10)	5.32(6)	4.04(1)	5.85(9)	5.82(8)	5.47(7)	9.60(15)	7.72(12)	9.03(14)	12.94(16)	12.95(17)	40.18(18)
125	4.44(3)	4.88(4)	8.00(13)	4.19(2)	4.90(5)	6.60(11)	5.50(9)	5.03(6)	3.83(1)	5.62(10)	5.48(8)	5.09(7)	9.36(15)	7.59(12)	8.89(14)	12.78(17)	12.59(16)	40.18(18)
150	4.21(3)	4.68(4)	7.77(13)	3.96(2)	4.73(5)	6.37(11)	5.21(8)	4.78(6)	3.65(1)	5.47(10)	5.29(9)	4.82(7)	9.23(15)	7.46(12)	8.80(14)	12.58(17)	12.34(16)	40.18(18)
175	4.02(3)	4.48(4)	7.57(13)	3.75(2)	4.55(6)	6.17(11)	4.94(8)	4.61(7)	3.44(1)	5.33(10)	5.08(9)	4.55(5)	9.14(15)	7.31(12)	8.75(14)	12.47(17)	12.18(16)	40.18(18)
200	3.85(3)	4.30(4)	7.42(13)	3.57(2)	4.41(7)	6.02(11)	4.72(8)	4.41(6)	3.30(1)	5.21(10)	4.88(9)	4.32(5)	9.01(15)	7.16(12)	8.70(14)	12.36(17)	11.98(16)	40.18(18)
225	3.68(3)	4.18(5)	7.28(13)	3.43(2)	4.30(7)	5.84(11)	4.53(8)	4.25(6)	3.19(1)	5.09(10)	4.74(9)	4.12(4)	8.86(15)	7.06(12)	8.65(14)	12.25(17)	11.79(16)	40.18(18)
250	3.56(3)	4.05(5)	7.19(13)	3.32(2)	4.17(7)	5.70(11)	4.37(8)	4.13(6)	3.09(1)	4.99(10)	4.60(9)	3.97(4)	8.77(15)	6.96(12)	8.64(14)	12.18(17)	11.66(16)	40.18(18)
275	3.44(3)	3.93(5)	7.08(13)	3.19(2)	4.05(7)	5.57(11)	4.21(8)	4.00(6)	2.97(1)	4.90(10)	4.50(9)	3.85(4)	8.70(15)	6.89(12)	8.62(14)	12.10(17)	11.56(16)	40.18(18)
300	3.35(3)	3.84(5)	6.97(13)	3.11(2)	3.95(7)	5.48(11)	4.05(8)	3.90(6)	2.86(1)	4.83(10)	4.42(9)	3.71(4)	8.61(14)	6.84(12)	8.62(15)	12.03(17)	11.47(16)	40.18(18)

Table 7. Average RPI values and their ranks (in brackets) for the instances with 125% setup times

ρ	ILS _{MRLS}	ILS _{HLS}	ILS _{VNS}	IG _{MRLS}	IG _{HLS}	IG _{VNS}	IFFO	MBO	ABC	ILSN	VIG	ILSD	GA	ISA	nFOA	AIS	IAIS	IA
25	5.52(3)	5.84(4)	10.09(13)	5.43(2)	6.18(6)	7.93(11)	6.84(8)	6.13(5)	4.75(1)	6.84(9)	7.15(10)	6.36(7)	11.28(15)	8.61(12)	10.41(14)	15.23(17)	14.63(16)	46.77(18)
50	5.21(3)	5.54(4)	9.77(13)	5.10(2)	5.83(6)	7.55(11)	6.41(8)	5.77(5)	4.48(1)	6.56(9)	6.72(10)	5.99(7)	11.10(15)	8.40(12)	10.28(14)	15.03(17)	14.25(16)	46.77(18)
75	4.87(3)	5.31(4)	9.52(13)	4.81(2)	5.59(7)	7.35(11)	6.01(8)	5.54(6)	4.25(1)	6.36(9)	6.39(10)	5.54(5)	10.92(15)	8.26(12)	10.18(14)	14.76(17)	13.89(16)	46.77(18)
100	4.64(3)	5.07(4)	9.34(13)	4.58(2)	5.41(7)	7.15(11)	5.70(8)	5.30(6)	4.05(1)	6.16(10)	6.14(9)	5.23(5)	10.78(15)	8.15(12)	10.12(14)	14.59(17)	13.65(16)	46.77(18)
125	4.44(3)	4.93(4)	9.16(13)	4.38(2)	5.19(7)	6.97(11)	5.38(8)	5.12(6)	3.86(1)	6.06(10)	5.92(9)	5.04(5)	10.66(15)	8.02(12)	10.05(14)	14.46(17)	13.41(16)	46.77(18)
150	4.26(3)	4.81(4)	9.00(13)	4.22(2)	5.07(7)	6.83(11)	5.16(8)	4.97(6)	3.72(1)	5.93(10)	5.74(9)	4.86(5)	10.55(15)	7.93(12)	10.03(14)	14.35(17)	13.18(16)	46.77(18)
175	4.10(3)	4.70(5)	8.89(13)	4.05(2)	4.95(8)	6.70(11)	4.95(7)	4.86(6)	3.60(1)	5.79(10)	5.54(9)	4.66(4)	10.45(15)	7.85(12)	10.01(14)	14.24(17)	13.02(16)	46.76(18)
200	3.98(3)	4.58(5)	8.79(13)	3.92(2)	4.86(8)	6.59(11)	4.79(7)	4.69(6)	3.50(1)	5.70(10)	5.38(9)	4.54(4)	10.40(15)	7.74(12)	9.98(14)	14.09(17)	12.87(16)	46.76(18)
225	3.84(3)	4.46(5)	8.70(13)	3.80(2)	4.73(8)	6.47(11)	4.60(7)	4.58(6)	3.42(1)	5.62(10)	5.27(9)	4.40(4)	10.32(15)	7.65(12)	9.97(14)	14.01(17)	12.76(16)	46.76(18)
250	3.56(3)	4.05(5)	7.19(13)	3.32(2)	4.17(7)	5.70(11)	4.37(8)	4.13(6)	3.09(1)	4.99(10)	4.60(9)	3.97(4)	8.77(15)	6.96(12)	8.64(14)	12.18(17)	11.66(16)	40.18(18)
275	3.44(3)	3.93(5)	7.08(13)	3.19(2)	4.05(7)	5.57(11)	4.21(8)	4.00(6)	2.97(1)	4.90(10)	4.50(9)	3.85(4)	8.70(15)	6.89(12)	8.62(14)	12.10(17)	11.56(16)	40.18(18)
300	3.35(3)	3.84(5)	6.97(13)	3.11(2)	3.95(7)	5.48(11)	4.05(8)	3.90(6)	2.86(1)	4.83(10)	4.42(9)	3.71(4)	8.61(14)	6.84(12)	8.62(15)	12.03(17)	11.47(16)	40.18(18)

Table 8. Best-known solutions for the benchmark instances [32] with 20 jobs (The new best-known solutions found by DABC are in bold)

instances	P13				P3			
	SSD10	SSD50	SSD100	SSD125	SSD10	SSD50	SSD100	SSD125
ta002	1025	1187	1376	1468	351	455	556	599
ta004	1178	1245	1338	1386	353	455	565	610
ta006	1314	1392	1557	1635	345	440	535	569
ta008	1067	1146	1230	1281	337	429	531	586
ta010	1031	1165	1325	1386	305	401	502	549
ta012	1439	1567	1818	1983	533	647	794	858
ta014	1325	1508	1704	1790	448	580	717	755
ta016	1344	1515	1804	1945	507	636	769	827
ta018	1408	1642	1962	2118	512	617	757	831
ta020	1315	1477	1677	1795	530	646	794	849
ta022	1889	2193	2571	2750	769	928	1108	1202
ta024	1845	2088	2471	2652	766	920	1112	1201
ta026	1845	2115	2474	2665	827	995	1168	1257
ta028	1842	2137	2490	2699	845	1002	1202	1289
ta030	1704	1977	2353	2521	856	1012	1185	1262

Table 9. Best-known solutions for the benchmark instances [32] with 50 jobs (The new best-known solutions found by DABC are in bold)

instances	P13				P3			
	SSD10	SSD50	SSD100	SSD125	SSD10	SSD50	SSD100	SSD125
ta032	2579	2810	3099	3277	660	919	1163	1304
ta034	2718	3020	3406	3608	655	934	1189	1300
ta036	2749	3201	3802	4062	657	907	1159	1258
ta038	2681	2923	3263	3432	619	901	1170	1302
ta040	2679	2956	3295	3485	665	900	1189	1295
ta042	2971	3409	3977	4327	826	1158	1501	1660
ta044	2960	3595	4334	4697	820	1139	1516	1663
ta046	2909	3463	4130	4446	900	1230	1570	1725
ta048	3002	3525	4126	4508	912	1225	1572	1719
ta050	2967	3347	3820	4029	876	1203	1538	1705
ta052	3314	4030	4894	5329	1198	1585	2023	2221
ta054	3178	3889	4718	5228	1248	1634	2068	2255
ta056	3302	4004	4844	5272	1270	1661	2068	2271
ta058	3421	4094	5051	5512	1218	1586	2035	2244
ta060	3400	4032	4842	5231	1262	1641	2082	2272

Fig. 1. Gantt diagram for a solution to the example problem

```

Procedure  $HLS(\pi, \beta, loop_{HLS})$ 
   $Cnt \leftarrow 1$ 
  Repeat
    If  $rand() < \beta$ 
       $\pi' \leftarrow Insert(\pi)$ 
    Else
       $\pi' \leftarrow Interchange(\pi)$ 
    Endif
    If  $C_{max}(\pi') < C_{max}(\pi)$ 
       $\pi \leftarrow \pi'$ 
    Endif
     $Cnt \leftarrow Cnt + 1$ 
  Until  $Cnt = loop_{HLS}$ 
Return  $\pi$ 

```

Fig. 2. Hybrid local search procedure

```

Procedure  $VNS(\pi, loop_{VNS})$ 
   $loop \leftarrow 1$ 
  Repeat
     $Cnt \leftarrow 1$ 
    Repeat
      If  $Cnt = 1$ 
         $\pi' \leftarrow Insert(\pi)$ 
      Elseif  $Cnt = 2$ 
         $\pi' \leftarrow Interchange(\pi)$ 
      Endif
      If  $C_{max}(\pi') < C_{max}(\pi)$ 
         $\pi \leftarrow \pi', Cnt \leftarrow 1$ 
      Elseif
         $Cnt \leftarrow Cnt + 1$ 
      Endif
    Until  $Cnt > 2$ 
     $loop \leftarrow loop + 1$ 
  Until  $loop = loop_{VNS}$ 
Return  $\pi$ 

```

Fig. 3. Variable neighborhood search

```

Procedure  $RLS_{insert}(\pi, \pi^s)$ 
   $i \leftarrow 1, Cnt \leftarrow 0$ 
  Repeat
    Find and extract job  $\pi_i^s$  from  $\pi$ 
    Test  $\pi_i^s$  in all possible insertion positions of  $\pi$ 
    Insert  $\pi_i^s$  at the position resulting in the lowest makespan, and denote the obtained sequence
     $\pi'$ 
    If  $C_{\max}(\pi') < C_{\max}(\pi)$ 
       $\pi \leftarrow \pi', Cnt \leftarrow 1$ 
    Else
       $Cnt \leftarrow Cnt + 1$ 
    Endif
     $i \leftarrow \text{mod}(i + 1, n)$ 
  Until  $Cnt = n$ 
Return  $\pi$ 

```

Fig. 4. Referenced local search with insertion

```

Procedure  $RLS_{Interchange}(\pi, \pi^s)$ 
   $i \leftarrow 1, Cnt \leftarrow 0$ 
  Repeat
    Find job  $\pi_i^s$  from  $\pi$ 
    Test to interchange  $\pi_i^s$  with all of the jobs before it in  $\pi$ 
    Interchange  $\pi_i^s$  with the job resulting in the lowest makespan, and denote the obtained sequence
     $\pi'$ 
    If  $C_{\max}(\pi') < C_{\max}(\pi)$ 
       $\pi \leftarrow \pi', Cnt \leftarrow 1$ 
    Else
       $Cnt \leftarrow Cnt + 1$ 
    Endif
     $i \leftarrow \text{mod}(i + 1, n)$ 
  Until  $Cnt = n$ 
Return  $\pi$ 

```

Fig. 5. Referenced local search with interchange

```

Procedure  $MRLS(\pi, \pi^s, \alpha)$ 
  If  $\alpha = 1$ 
     $\pi \leftarrow RLS_{insert}(\pi, \pi^s, \alpha)$ 
  Elsif  $\alpha = 2$ 
     $\pi \leftarrow RLS_{Interchange}(\pi, \pi^s, \alpha)$ 
  Elseif  $\alpha = 3$ 
     $\pi \leftarrow RLS_{insert}(\pi, \pi^s, \alpha)$ 
     $\pi \leftarrow RLS_{Interchange}(\pi, \pi^s, \alpha)$ 
  Elseif  $\alpha = 4$ 
     $\pi \leftarrow RLS_{Interchange}(\pi, \pi^s, \alpha)$ 
     $\pi \leftarrow RLS_{insert}(\pi, \pi^s, \alpha)$ 
  Endif
Return  $\pi$ 

```

Fig. 6. Mixed referenced local search procedure

```

Procedure Perturbation( $\pi, \omega, \varpi$ )
   $\Upsilon = \{ \}$ 
  Repeat
     $\pi' \leftarrow$  perform  $\omega$  random insertions to  $\pi$ 
    Add  $\pi'$  to  $\Upsilon$ .
  Until  $|\Upsilon| = \varpi$ 
   $\pi \leftarrow$  best solution among set  $\Upsilon$ 
Return  $\pi$ 

```

Fig. 7. Perturbation procedure

```

Procedure SimulatedAnnealingCriterion( $\pi, \pi', T$ )
  If  $C_{\max}(\pi') \leq C_{\max}(\pi)$ 
     $\pi \leftarrow \pi'$ 
  Elseif  $\text{rand}() < \exp((C_{\max}(\pi) - C_{\max}(\pi'))/T)$ 
     $\pi \leftarrow \pi'$ 
  Endif
Return  $\pi$ 

```

Fig. 8. Acceptance criterion

```

Procedure ILSHLS
  Set parameters:  $\lambda$ ,  $\beta$ ,  $loop_{HLS}$ ,  $\omega$ , and  $\varpi$ .
   $T \leftarrow$  Get temperature by formulation (5)
   $\pi \leftarrow NEH$ 
   $\pi \leftarrow HLS(\pi, \beta, loop_{HLS})$ 
   $\pi^* \leftarrow \pi$ 
  Repeat
     $\pi' \leftarrow Perturbation(\pi, \omega, \varpi)$ 
     $\pi'' \leftarrow HLS(\pi', \beta, loop_{HLS})$ 
    If  $C_{\max}(\pi'') < C_{\max}(\pi^*)$ 
       $\pi^* \leftarrow \pi''$ 
    Endif
     $\pi \leftarrow SimulatedAnnealingCriterion(\pi, \pi'', T)$ 
  Until termination criterion is met
Return  $\pi^*$ 

```

Fig. 9. ILS_{HLS} procedure

```

Procedure ILSVNS
  Set parameters:  $\lambda$ ,  $loop_{VNS}$ ,  $\omega$ , and  $\varpi$ .
   $T \leftarrow$  Get temperature by formulation (5)
   $\pi \leftarrow NEH$ 
   $\pi \leftarrow VNS(\pi, loop_{VNS})$ 
   $\pi^* \leftarrow \pi$ 
  Repeat
     $\pi' \leftarrow Perturbation(\pi, \omega, \varpi)$ 
     $\pi'' \leftarrow VNS(\pi', loop_{VNS})$ 
    If  $C_{\max}(\pi'') < C_{\max}(\pi^*)$ 
       $\pi^* \leftarrow \pi''$ 
    Endif
     $\pi \leftarrow SimulatedAnnealingCriterion(\pi, \pi'', T)$ 

```

Until termination criterion is met
Return π^*

Fig. 10. ILS_{VNS} procedure

Procedure ILS_{MRLS}
Set parameters: λ , π^s , α , ω , and ϖ .
 $T \leftarrow$ Get temperature by formulation (5)
 $\pi \leftarrow NEH$
 $\pi \leftarrow MRLS(\pi, \pi^s, \alpha)$
 $\pi^* \leftarrow \pi$
Repeat
 $\pi' \leftarrow Perturbation(\pi, \omega, \varpi)$
 $\pi'' \leftarrow MRLS(\pi', \pi^s, \alpha)$
If $C_{max}(\pi'') < C_{max}(\pi^*)$
 $\pi^* \leftarrow \pi''$
Endif
 $\pi \leftarrow SimulatedAnnealingCriterion(\pi, \pi'', T)$
Until termination criterion is met
Return π^*

Fig. 11. ILS_{MRLS} procedure

Procedure $MDC(\pi, d, Enh)$
 $\pi^d = ()$
For $i = 1 : d$ //Destruction
Randomly extract a job from π
Append the job to π^d
Endfor
For $i = 1 : d$ //Construction
Extract the first job from π^d
Test the job at all possible slots of π
Insert the job in the best slot, supposing p , resulting in the lowest makespan
If $Enh = true$
Shift the job occupying position $p-1$ to its best slot
Shift the job occupying position $p+1$ to its best slot
Endif
Endfor
Return π

Fig. 12. MDC procedure

Procedure IG_{HLS}
Set parameters: λ , β , $loop_{HLS}$, d and Enh
 $T \leftarrow$ Get temperature by formulation (5)
 $\pi \leftarrow NEH$
 $\pi \leftarrow HLS(\pi, \beta, loop_{HLS})$
 $\pi^* \leftarrow \pi$
Repeat
 $\pi' \leftarrow MDC(\pi, d, Enh)$
 $\pi'' \leftarrow HLS(\pi', \beta, loop_{HLS})$
If $C_{max}(\pi'') < C_{max}(\pi^*)$
 $\pi^* \leftarrow \pi''$

```

Endif
 $\pi \leftarrow \text{SimulatedAnnealingCriterion}(\pi, \pi'', T)$ 
Until termination criterion is met
Return  $\pi^*$ 

```

Fig. 13. IG_{HLS} procedure

```

Procedure  $IG_{VNS}$ 
Set parameters:  $\lambda$ ,  $loop_{VNS}$ ,  $d$  and  $Enh$ 
 $T \leftarrow$  Get temperature by formulation (5)
 $\pi \leftarrow NEH$ 
 $\pi \leftarrow VNS(\pi, loop_{VNS})$ 
 $\pi^* \leftarrow \pi$ 
Repeat
 $\pi' \leftarrow MDC(\pi, d, Enh)$ 
 $\pi'' \leftarrow VNS(\pi', loop_{VNS})$ 
If  $C_{\max}(\pi'') < C_{\max}(\pi^*)$ 
 $\pi^* \leftarrow \pi''$ 
Endif
 $\pi \leftarrow \text{SimulatedAnnealingCriterion}(\pi, \pi'', T)$ 
Until termination criterion is met
Return  $\pi^*$ 

```

Fig. 14. IG_{VNS} procedure

```

Procedure  $IG_{MRLS}$ 
Set parameters:  $\lambda$ ,  $\pi^s$ ,  $\alpha$ ,  $d$  and  $Enh$ 
 $T \leftarrow$  Get temperature by formulation (5)
 $\pi \leftarrow NEH$ 
 $\pi \leftarrow MRLS(\pi, \pi^s, \alpha)$ 
 $\pi^* \leftarrow \pi$ 
Repeat
 $\pi' \leftarrow MDC(\pi, d, Enh)$ 
 $\pi'' \leftarrow MRLS(\pi', \pi^s, \alpha)$ 
If  $C_{\max}(\pi'') < C_{\max}(\pi^*)$ 
 $\pi^* \leftarrow \pi''$ 
Endif
 $\pi \leftarrow \text{SimulatedAnnealingCriterion}(\pi, \pi'', T)$ 
Until termination criterion is met
Return  $\pi^*$ 

```

Fig. 15. IG_{MRLS} procedure

```

Procedure PathRelinking( $\pi''$ )
 $\pi \leftarrow$  Select a target solution using the methods in Section 4.1.3
Set an intermediate solution  $\pi' \leftarrow \pi''$ 
Set the intermediate solution set  $\Gamma \leftarrow \{ \}$ 
While  $\pi' \neq \pi$  do
randomly select a position  $k$  such that  $\pi'_k \neq \pi_k$ 
find the position of job  $\pi_k$  on permutation  $\pi'$ 
interchange the positions of the pair of jobs  $(\pi'_k, \pi_k)$  on permutation  $\pi'$ 
 $\Gamma \leftarrow \Gamma \cup \{\pi'\}$  if  $\pi' \neq \pi$ 
End while
exclude the first solution in  $\Gamma$  if it is not better than starting solution  $\pi''$ 
exclude the last solution in  $\Gamma$  if it is not better than target solution  $\pi$ 
If  $\Gamma \neq \{ \}$ 

```



```

 $\pi' \leftarrow$  the best solution in  $\Gamma$ 
Else
 $\pi' \leftarrow$  perform two rand exchange moves to  $\pi''$ 
End if
perform the exploration and exploitation search (Section 4.1.2) to  $\pi'$ 
If  $C_{\max}(\pi') < C_{\max}(\pi'')$ 
 $\pi'' \leftarrow \pi'$ 
End if
Return  $\pi''$ 

```

Fig. 16. Path relinking based cooperative search

```

Procedure DiversityControl( $\Pi$ )
Calculate job block matrix  $[\delta_{j',j}]_{n \times n}$ 
For each  $\pi \in \Pi$ 
If  $\pi$  belongs to the best 50% solutions
Calculate the remaining probability  $\mu_k$  for each job  $\pi_k$  in  $\pi$ 
Extract jobs from  $\pi$  according to the remaining probability
Randomly insert missing jobs into  $\pi$ 
Else
 $\pi \leftarrow$  Randomly generate a new solution
End if
End for
Return  $\Pi$ 

```

Fig. 17. Diversity control method

```

Procedure IFFO
Set  $NS$ ,  $\mathcal{G}$ , and  $\gamma_{\max}$ 
Initialize populations  $\Pi \leftarrow \{\Pi_1, \Pi_2, \dots, \Pi_{NS}\}$  using SPT-based heuristic
 $\pi^* \leftarrow \arg\left(\min_{k=1,2,\dots,NS} (C_{\max}(\Pi_k))\right)$ 
 $\gamma \leftarrow 0$ 
Repeat
For  $k=1,2,\dots,NS$  do
//Exploration phase
Randomly generate  $\mathcal{G}$  solutions using the hybrid neighboring operator for  $\Pi_k$ 
 $\pi \leftarrow$  The best one among the generated  $\mathcal{G}$  solutions

//Exploitation phase
If  $C_{\max}(\pi) < C_{\max}(\Pi_k)$ 
 $\Pi_k \leftarrow \pi$ 
Endif
If  $C_{\max}(\pi) < C_{\max}(\pi^*)$ 
 $\pi^* \leftarrow \pi$ ,  $\gamma \leftarrow 0$ 
Endif
Endfor

//Path relinking based cooperative search
For  $k=1,2,\dots,NS$  do
 $\Pi_k \leftarrow \text{PathRelinking}(\Pi_k)$ 
If  $C_{\max}(\Pi_k) < C_{\max}(\pi^*)$ 
 $\pi^* \leftarrow \pi$ ,  $\gamma \leftarrow 0$ 
Endif
Endfor

 $\gamma \leftarrow \gamma + 1$ 

// Diversity control
If  $\gamma = \gamma_{\max}$ 

```

```

 $\Pi \leftarrow \text{DiversityControl}(\Pi)$ 
 $\gamma \leftarrow 0$ 
End if
Until a termination criterion is met
Return  $\pi^*$ 

```

Fig. 18. IFFO procedure

Fig. 19. Means plots of the parameters and operators of IFFO

Fig. 20. Means plots of the competing algorithms for the instances with 25% setup times

Fig. 21. Means plots of the competing algorithms for the instances with 50% setup times

Fig. 22. Means plots of the competing algorithms for the instances with 100% setup times

Fig. 23. Means plots of the competing algorithms for the instances with 125% setup times