Wang, Y.; Li, X.; Ruiz García, R.; Sui, S. (2018). An Iterated Greedy Heuristic for Mixed No-Wait Flowshop Problems. IEEE Transactions on Cybernetics. 48(5):1553-1566. https://doi.org/10.1109/TCYB.2017.2707067

An Iterated Greedy Heuristic for Mixed No-wait Flowshop Problems

Yamin Wang, Xiaoping Li, *Senior Member, IEEE,* Rubén Ruiz and Shaochun Sui

***Abstract*—The mixed no-wait flowshop problem with both wait and no-wait constraints has many potential real-life applications. The problem can be regarded as a generalization of the traditional permutation flowshop and the no-wait flowshop. In this paper, we study, for the first time, this scheduling setting with makespan minimization. We first propose a mathematical model and then we design a speed-up makespan calculation procedure. By introducing a varying number of destructed jobs, a modified iterated greedy algorithm is proposed for the considered problem which consists of four components: initialization solution construction, destruction, reconstruction and local search. To further improve the intensification and efficiency of the proposal, insertion is performed on some neighbor jobs of the best position in a sequence during the initialization, solution construction and reconstruction phases. After calibrating parameters and components, the proposal is compared with five existing algorithms for similar problems on adapted Taillard benchmark instances. Experimental results show that the proposal always obtains the best performance among the compared methods.**

***Index Terms*—Flowshop, No-wait, Heuristics, Iterated greedy.**

I. Introduction

The permutation flowshop problem (PFSP) is one of the most well studied scheduling problems. A set of n jobs have to be processed on m machines. Each job passes through the machines following the same route. Each machine processes jobs in the same order, i.e., all machines have the same permutation of jobs. The problem is to find a job sequence optimizing one or more objectives. The no-wait flowshop problem (NWFSP), a constrained PFSP variant, has also been studied widely due to its applications in many industries, e.g., chemical processing, food processing, plastic molding and steel rolling [1]. In the NWFSP, waiting is not permitted, i.e., a job has to be processed continuously through the m machines without interruption once it starts on the first machine. When needed, the start of a job on the first machine is delayed in order to meet the no-wait requirement. This no-wait constraint models many real-life situations such as the need to process steel products while they are hot or processing frozen goods before they thaw being two examples.

Yamin Wang and Xiaoping Li are with the School of Computer Science and Engineering, Southeast University, Nanjing 211189, China, and also with Key Laboratory of Computer Network and Information Integration, Ministry of Education, Nanjing, 211189, China (e-mail: {wangyamin,xpli}@seu.edu.cn).

Rubén Ruiz is with Grupo de Sistemas de Optimización Aplicada, Instituto Tecnológico de Informática, Ciudad Politécnica de la Innovación, Edifico 8G, Acc. B. Universitat Politècnica de València, Camino de Vera s/n, 46021, València, Spain (e-mail: rruiz@eio.upv.es).

Shaochun Sui is with the Production Management, AVIC Chengdu Aircraft Industrial (Group) CO., LTD., Chengdu 610091, China (e-mail: suishaochun@vip.163.com).

Similarly to other scheduling problems [2]–[6], wait and no-wait constraints might co-exist at the same time in many real applications. For example, in the canned food processing industry, no-wait is not needed for many operations such as purchasing, classification, pruning, cleaning and removing the peel and shell. On the contrary, no-wait is required for the following operations: adding sugar liquid, gas exhausting, sealing, sterilizing and refrigerating once the food is precooked and while it is still hot. As a result the no-wait constraint is not needed again for subsequent operations such as the labeling, handling, palletizing, etc. because the food has been preserved safely in cans. Another typical example is producing mannitol from starch. Once size-mixing starts, no wait operations follow immediately (the first jet liquifying, liquifying in a reaction jar, the second jet liquifying, refrigerating, adjusting PH value and saccharifying). Any wait in between two operations would result in the starch becoming thick and then solid after it is heated and size mixed. However, waiting is permitted in later operations such as concentrating, separating and crystallizing. There are many similar examples in real manufacturing industries. These kinds of mixed flowshop scheduling problems (which are called MWFSPs in this paper) are different from PFSPs and NWFSPs and share all of the difficulties and some of the properties of both the regular wait PFSP and the NWFSP. The MWFSP with makespan criterion is denoted as $F_m|mixed, no-wait|C_{\max}$ using the 3-tuple notation by Graham et al. [7]. In fact, the MWFSP is a generalization of the PFSP and NWFSP. MWFSPs are PFSPs if there is no-wait constraint and they become NWFSPs if all jobs are no-wait constrained on all machines. Because of the NP-hard characteristic of both the PFSP [8] and NWFSP [9] when $m > 2$, it follows that the MWFSP is NP-hard for more than 2-machines. To the best of our knowledge, this problem has never been studied in the literature.

State-of-the-art methods for flowshop problems are highly effective and efficient. The NWFSP with makespan minimization can be reduced to the traveling salesman problem [10]. The distance between any adjacent jobs on the last machine is a constant and is determined by the processing times of the jobs, i.e., the distance remains unchanged no matter where the pair of jobs are located [11]–[13]. According to this property, the time complexity of makespan calculation can be reduced from $O(mn)$ to $O(n)$ [1], [14], which leads to a substantial increase in efficiency. For example, the time complexity of the insertion neighborhood search is $O(n^2)$ for the NWFSP while it is $O(mn^2)$ for the PFSP, i.e., the insertion neighborhood search is more efficient for the NWFSP than for the PFSP. However, obtaining the same efficiency for the MWFSP is a

challenge because jobs are only no-wait constrained on some machines.

The main contributions of this paper are summarized as follows:

- The considered MWFSP is mathematically modelled using 0-1 integer programming.
- We construct an accelerated makespan calculation method with time complexity $O((\xi+q)n)$ where ξ is the number of machines without the no-wait constraint and q is the number of no-wait groups for the remaining $m-\xi$ machines.
- A modified iterated greedy algorithm where the number of jobs to destroy is dynamically set is presented for the MWFSP under study. New neighborhood structures are constructed in terms of which VND-based local search methods are developed.

The rest of the paper is organized as follows: Related works are described in Section II. Section III details the considered problem and presents a mathematical model. Accelerated makespan calculation and speed-up neighborhood search methods are given in Section IV. Section V contains the details of the modified iterated greedy algorithm for the MWFSP. Experimental results of parameter calibration and algorithm comparison are shown in Section VI, followed by conclusions and future research in Section VII.

II. Related work

Even though the MWFSP has not been studied yet, it is a generalization of both the PFSP and the NWFSP. We pay more attention to the NWFSP because the PFSP has been well studied . The 2-machine NWFSP is identical to the 2-machine PFSP [15] which implies that $F_2|no-wait|C_{\max}$ can be optimally solved using the Johnson method [16]. Kalczynski et al. [10] reduced the $F_m|prmu,no{-}wait|C_{\max}$ to the traveling salesman problem. Two commonly adopted methods for the NWFSP are heuristics and metaheuristics.

Nawaz et al. proposed the NEH [17] algorithm which is an effective constructive heuristic widely used for the PFSP. Based on the NEH, many variants have been developed with different strategies and initial job orderings or seeds. Haupt and Reinhard [18] and Ramasesh [19] generated seeds by sorting jobs in the descending order of the sums of their processing times, i.e., the LPT (the longest processing time) rule. Pan et al. [14] found that the SPT (the shortest processing time) rule which arranges jobs in the ascending order of the sums of their processing times is effective for $F_m|prmu,no-wait|C_{\max}$. Ding et al. [20] generated an initial sequence by arranging jobs in non-increasing order of their standard deviations (STD). Sapkal and Laha [21] presented an efficient heuristic method for $F_m|prmu,no-wait|TFT$ which generated the initial sequence of jobs based on bottleneck machines. Edy Bertolissi [22] proposed a heuristic for $F_m|prmu,no-wait|TFT$ constructed the seed by selecting the smallest pair of partial flow-times (e.g. $F_m(pq)$ and $F_m(qp)$) and marking the starting job of the pair. The seed was created by ordering jobs in decreasing order of the marks. For $F_m|prmu,no{-}wait|\sum T_i$, Liu et al. [23] proposed six heuristics: two dispatching rules (SPT, EDD), three simple constructive heuristics (SLACK, SLACKRW and MDD) and a modified NEH algorithm which generates the initial job sequence by sorting jobs by the EDD rule. Experimental results showed that the modified NEH was the best.

Meta-heuristic algorithms are effective for the NWFSP which is known to be NP-hard when the number of machines is more than two. Pan et al. [24] proposed a hybrid discrete particle swarm optimization algorithm (HDPSO) for the no-wait flow shop scheduling problem with makespan criterion which outperforms both the single discrete particle swarm optimization algorithm (DPSO) and the hybrid particle swarm optimization (HPSO) algorithm in searching quality, robustness and efficiency. Pan et al. [1] presented a discrete particle swarm optimization (DPSO_{VND}) algorithm for the NWFSP considering both makespan and total flow time criteria. It was hybridized with the variable neighborhood descent (VND) algorithm to further improve the quality of solutions. Several speed-up methods were developed for both swap and insert neighborhood structures. Tseng and Lin [25] and Jarboui et al. [26] presented hybrid genetic algorithms (GA) respectively. The insertion search (IS) was used for a small neighborhood and the insert search with cut-and-repair (ISCR) searched a large neighborhood in the algorithm presented in [25]. Jarboui, Eddaly and Siarry [26] adopted the variable neighborhood search for improvement in the last step. Wang and Li [12] proposed an accelerated tabu search (TS) to minimize the maximum lateness. AitZai et al. [27] investigated a branch-and-bound algorithm and a particle swarm optimization (PSO) algorithm. Pan and Wang [14] proposed an improved iterated greedy algorithm (IIGA) for $F_m|prmu,no{-}wait|C_{\max}$. Computational results showed that the IIGA was more effective and efficient than TS, TS+M, TS+MP [28], and DPSO. Ding et al. [20] proposed an improved iterated greedy algorithm with a Tabu-based reconstruction strategy (TMIIG) for $F_m|prmu,no{-}wait|C_{\max}$. TMIIG performs better compared to other effective algorithms, such as IIGA [14], $DPSO_{VND}$ [1], GA_{VNS} [26] for the no-wait flowshop scheduling problem with a makespan criterion. The above analysis indicates that iterated greedy algorithms are always effective and efficient for flowshop scheduling problems with the no-wait constraint. This leads us to develop iterated greedy based algorithms for the problem under study: a mixed no-wait flowshop problem.

III. Problem description and mathematical model

The mixed no-wait flowshop problem (MNWFS for short) is described as follows: a set of n jobs $\mathbb{J}=\{J_i|\, i=1,\,2,\cdots,\,n\}$ are available at time zero. They have to be processed on m machines $\mathbb{M}=\{M_j|j=1,\,2,\cdots,\,m\}$ sequentially. Each machine processes all jobs in the same order. No machine processes more than one job at a time, job preemption is not allowed and setup times are included in job processing times. The operation $O_{i,j}$ of job J_i on machine M_j ($i=1,\,2,\cdots,\,n,\ j=1,\,2,\cdots,\,m$) is processed without interruption, of which the processing time is $p_{i,j}$ ($p_{i,j}\geq 0$). Unlike the NWFSP problem, in which all operations of each job are no-wait constrained, only some MNWFS operations

are no-wait constrained and the others are traditional PFSP operations. That is to say, no-wait machines are grouped into sub-sets by regular (non no-wait) machines. Let $\mathbb{M}^w$ be the set of regular machines and $\xi = |\mathbb{M}^w|$. $\bigcup_{i=1}^{q} \mathbb{M}^i$ denotes the set of no-wait machines where $\mathbb{M}^i$ is the i^{th} group of no-wait machines and q is the number of no-wait machine groups. It is obvious that $\mathbb{M}^i \bigcap \mathbb{M}^j = \varnothing$ $(i \neq j)$. No waiting time is permitted for processing any job between consecutive operations on machines in $\mathbb{M}^i$ $(i = 1, 2, \cdots, q)$. Operations on machines $\mathbb{M}^w$ have no such constraint. For example, assume M_3, M_4, M_5, M_8 and M_9 are no-wait machines in a 10-machine MNWFS problem. There are two groups of no-wait machines $\mathbb{M}^1 = \{M_3, M_4, M_5\}$ and $\mathbb{M}^2 = \{M_8, M_9\}$. $\mathbb{M}^w = \{M_1, M_2, M_6, M_7, M_{10}\}$. The target is to find the optimum schedule of the n jobs with the minimum maximum completion time or makespan ($C_{\max}$). Notations used in this paper are shown in Table I.

TABLE I
NOTATION EMPLOYED IN THIS PAPER

n	Number of jobs
m	Number of machines
$\mathbb{J}$	Set of n jobs to be processed
J_i	the i^{th} job in $\mathbb{J}$, $i = 0, 1, 2, \cdots, n$
M_j	the j^{th} machine, $j = 0, 1, 2, \cdots, m$
$O_{i,j}$	Operation of job J_i on machine M_j
$p_{i,j}$	Processing time of $O_{i,j}$
$\mathbb{M}^w$	Set of regular machines
ξ	Number of regular machines, i.e., $\xi = \|\mathbb{M}^w\|$
M^i	the i^{th} group of no-wait machines
q	Number of no-wait machine groups
$C_{\max}$	Minimum maximum completion time or makespan
$S_{k,j}$	Start time of the job at the k^{th} position of job sequence π on machine M_j
$C_{k,j}$	Completion time of the job at the k^{th} position of job sequence π on machine M_j
$d^1_{i,j,r}$	The minimum delay between the completion times of J_i and the start times of J_j on the first machine of the group $\mathbb{M}^r$ $(r = 1, 2, \cdots, q)$ where J_i is located before J_j in a sequence.
$d^2_{i,j,r}$	The minimum difference between the completion times of the pair of jobs on the last machine of every $\mathbb{M}^r$ $(r = 1, 2, \cdots, q)$.

For simplicity, a dummy job J_0 is introduced at the start of a schedule and a dummy machine M_0 is used at the start of each job. Operation processing times of J_0 are zero and those of all jobs on M_0 are zero as well, i.e., $p_{0,j} = 0$ $(j = 0, 1, 2, \cdots, m)$ and $p_{i,0} = 0$ $(i = 0, 1, 2, \cdots, n)$. Decision variables $x_{i,k}$ $(i,\ k \in \{1,\ 2, \cdots, n\})$ are introduced. $x_{i,k} = 1$ if J_i is located at the k^{th} position of a sequence; otherwise, $x_{i,k} = 0$. We suppose there are l_i machines in the $\mathbb{M}^i$ no-wait group (i.e., $\sum_{i=1}^{q} l_i = m - \xi$). $\mathbb{M}^i$ is denoted as $\{\mathbb{M}^i_{[1]}, \ldots, \mathbb{M}^i_{[l_i]}\}$ $(i = 1, \ldots, q)$ and $\mathbb{M}^a = \{\mathbb{M}^1_{[1]}, \mathbb{M}^2_{[1]}, \cdots, \mathbb{M}^q_{[1]}\}$. Let π be a permutation of n jobs $\pi = (\pi_{[0]}, \pi_{[1]}, \pi_{[2]}, \cdots, \pi_{[n]})$. $S_{k,j}$ denotes the start time of the job at the k^{th} position of π on machine M_j. $C_{k,j}$ represents the completion time of the job at the k^{th} position on machine M_j. Obviously, $S_{0,j} = C_{0,j} = 0$ $(j = 0, 1, 2, \ldots, m)$. The problem under study is mathematically modeled as follows.

$$\min C_{\max} = C_{n,m} \tag{1}$$

$$\text{s.t.} \quad \sum_{k=1}^{n} x_{i,k} = 1, \quad i = 1, 2, \cdots, n \tag{2}$$

$$\sum_{i=1}^{n} x_{i,k} = 1, \quad k = 1, 2, \cdots, n \tag{3}$$

$$C_{k,j} \geq C_{k-1,j} + \sum_{i=1}^{n} x_{i,k} p_{i,j}, \quad j = 1, 2, \cdots, m;\ k = 1, 2, \cdots, n \tag{4}$$

$$C_{k,j} \geq C_{k,j-1} + \sum_{i=1}^{n} x_{i,k} p_{i,j}, \quad \forall M_j \in \mathbb{M}^w \bigcup \mathbb{M}^a \tag{5}$$

$$C_{k,j} = C_{k,j-1} + \sum_{i=1}^{n} x_{i,k} p_{i,j}, \quad \forall M_j \in \mathbb{M} - \mathbb{M}^w - \mathbb{M}^a \tag{6}$$

$$x_{i,k} \in \{0,\ 1\} \quad i = 1,\ 2, \cdots, n,\ k = 1,\ 2, \cdots, n \tag{7}$$

Equation (2) ensures that each job occurs only once in the permutation. Equation (3) guarantees that each position is occupied by only one job. Constraint (4) implies that a job can only start after the previous job on the same machine finishes. Constraint (5) indicates that an operation of a job starts only after its previous operation finishes. Equation (6) means that there is no waiting time between any two consecutive no-wait machines. Constraints (7) define the decision variables.

IV. SPEED-UP METHODS

A highly efficient makespan calculation is crucial for efficiency in flowshop problems. This is no different for the MWFSP. We can calculate the makespan in a traditional way, without using accelerations for the MWFSP as shown in Algorithm 1. Completion times of jobs are calculated one by one. Completion times of each job on machines are firstly calculated as in the regular PFSP. Then completion times on no-wait machines are adjusted using a backward shift according to the no-wait constraint. The time complexity the adjustment for each job is $O(m)$. Therefore, the time complexity of Algorithm 1 is $O(mn)$.

ALGORITHM 1: General Makespan Calculation (GMC)

```
1  begin
2      for j = 0 to n do
3          C_{0,j} = 0;
4      for k = 1 to n do
5          C_{k,0} = 0;
6          for j = 1 to m do
7              S_{k,j} = max(C_{k,j-1}, C_{k-1,j}),
               C_{k,j} = S_{k,j} + p_{π_[k],j} ;
8          for u = q to 1 do
9              for j = M^u_{[l_u - 1]} to M^u_{[1]} do
10                 C_{k,j} = S_{k,j+1}, S_{k,j} = C_{k,j} - p_{π_[k],j};
11     return C_{n,m}.
```

Since there are some groups of no-wait machines, it is possible to improve the efficiency of makespan calculation as in Li et al. [13]. The distance of any pair of jobs on

each machine in a no-wait machine group is constant for any schedule [10] [13], i.e., the distance is independent of the positions of adjacent pairs of jobs. In this paper, we obtain the makespan values of the schedules by calculating the completion times of each job on no-wait machines and those on the other machines separately. Every no-wait machine group is regarded separately and the size (number of machines) of the original problem is reduced.

We compute the distances of all pairs of jobs on every no-wait machine group in advance and keep them in matrix D. For each pair of jobs J_i and J_j $(i \neq j)$, $d^1_{i,j,r}$ and $d^2_{i,j,r}$ are computed. As shown in Figure 1, $d^1_{i,j,r}$ is the minimum delay between the completion times of J_i and the start times of J_j on the first machine of the group $\mathbb{M}^r$ $(r = 1, 2, \cdots, q)$ where J_i is located before J_j in a sequence. $d^2_{i,j,r}$ is the minimum difference between the completion times of the pair of jobs on the last machine of every $\mathbb{M}^r$ $(r = 1, 2, \cdots, q)$. $d^1_{i,j,r}$ and $d^2_{i,j,r}$ are calculated by Equations (8) and (9) respectively.

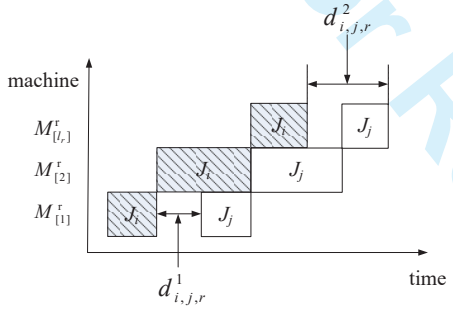

Fig. 1. $d^1_{i,j,r}$ and $d^2_{i,j,r}$

$$d^2_{i,j,r} = \max_{h \in M^r} \left\{ \sum_{t=h}^{\mathbb{M}^r_{[l_r]}} (p_{j,t} - p_{i,t}) + p_{i,h} \right\} \tag{8}$$

$$d^1_{i,j,r} = d^2_{i,j,r} + \sum_{h=\mathbb{M}^r_{[2]}}^{\mathbb{M}^r_{[l_r]}} p_{i,h} - \sum_{h=\mathbb{M}^r_{[1]}}^{\mathbb{M}^r_{[l_r]}} p_{j,h} \tag{9}$$

The time complexity of computing matrix D is $O(n^2(m-\xi))$. Even though it is greater than that of the GMC (which is $O(mn)$), the obtained matrix only has to be calculated once and is available for the whole search process which can reduce the computation time greatly. Computation time comparisons will be given in the experimental section.

With the obtained matrix, the original problem is reduced to a pseudo-PFSP problem with $\xi + q$ machines, i.e., all q groups of no-wait machines are regarded as q artificial machines. Therefore, as jobs do not wait in between all these q machine groups we can consider all their tasks together in a single artificial machine. Computation times of jobs are calculated one by one. Assume job J_j is appended to a job sequence whose last job is J_i. When J_j is appended to the no-wait machine group $\mathbb{M}^r = \{\mathbb{M}^r_{[1]}, \mathbb{M}^r_{[2]}, \ldots, \mathbb{M}^r_{[l_r]}\}$ (in which $\mathbb{M}^r_{[1]} = M_k$), a shift $a = \max\{0,\ C_{k,j-1} - C_{k-1,j} - d^1_{\pi_{[k-1]},x,i}\}$ is performed to meet the $S_{j,k} \geq C_{j,k-1}$ requirement, which is different from the makespan computation process for traditional PFSP problems. J_j on the no-wait machine group $\mathbb{M}^r$ needs to be shifted right if $a > 0$. Figure 2 shows the $a = 0$ case and Figure 3 depicts the $a > 0$ case. In this paper, the makespan computation process is called Speed-up Makespan Calculation (SMC for short) which is formally described in Algorithm 2. Since the time complexity of appending a new job to a partial sequence is $O(\xi + q)$, it is obvious that the time complexity of SMC is $O((\xi + q)n)$ which is much more efficient than GMC if both ξ and q are small.

ALGORITHM 2: Speed-up Makespan Calculation (SMC)

```
1  begin
2    for j = 0 to m do
3      C_{0,j} = 0;
4    for k = 1 to n do
5      C_{k,0} = 0, v = 1, j = 1;
6      while j ≤ m do
7        if M_j ∈ M^w then
8          S_{k,j} = max(C_{k,j-1}, C_{k-1,j}),
           C_{k,j} = S_{k,j} + p_{π[k],j}, j = j + 1;
9        if M_j = M^v_{[1]} then
10         if (C_{k,j-1} − C_{k-1,j} ≤ d^1_{π[k-1],π[k],v}) then
11           a = 0;
12         else
13           a = C_{k,j-1} − C_{k-1,j} − d^1_{π[k-1],π[k],v};
14         C_{k,j} = C_{k-1,j} + d^1_{π[k-1],π[k],v} + p_{π[k],j} + a;
15         C_{k,j+l_v-1} =
               C_{k-1,j+l_v-1} + d^2_{π[k-1],π[k],v} + a;
               // l_v is the number of
               machines in M^v
16         j = j + l_v, v = v + 1;
17   return C_{n,m}.
```

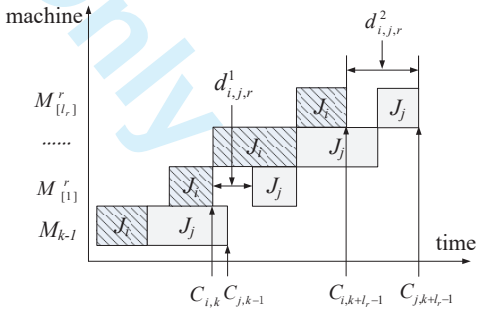

Fig. 2. The $a = 0$ case

To illustrate the SMC calculation process, we give an example with three jobs $J = \{J_1, J_2, J_3\}$ and four machines $M = \{M_1, M_2, M_3, M_4\}$. Assume M_2, M_3 and M_4 are no-wait machines which form one no-wait machine group $\mathbb{M}^1 = \{M_2, M_3, M_4\}$. Operation processing times of the jobs are the following:

$$[p_{i,j}]_{4\times 3} = \begin{bmatrix} 3 & 5 & 6 \\ 6 & 3 & 2 \\ 1 & 3 & 3 \\ 4 & 2 & 4 \end{bmatrix}$$

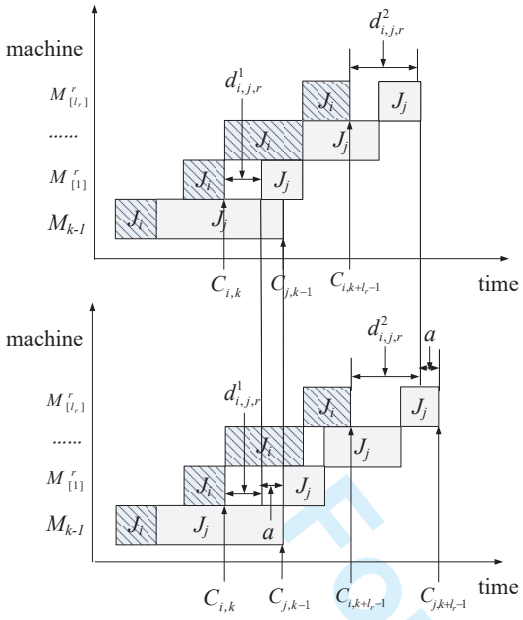

Fig. 3. The $a > 0$ case

The makespan of $\pi = (J_0, J_1, J_2, J_3)$ is calculated in the following way: (i) Matrix D is calculated in advance by Equations (8) and (9) in which $d^1_{0,1,1} = 0$, $d^2_{0,1,1} = 11$, $d^1_{1,2,1} = 0$, $d^2_{1,2,1} = 3$, $d^1_{2,3,1} = 1$, $d^2_{2,3,1} = 5$. (ii) SMC is called to calculate the makespan of π which is also illustrated in Figure 4.

- When π contains only job J_0, completion times $C_{0,j}$ $(j = 0, 1, \cdots, 4)$ are assigned to 0.
- J_1 is appended to π, i.e., $\pi = (J_0, J_1)$. $C_{1,0} = 0$. Because $S_{1,1} = \max(C_{1,0}, C_{0,1}) = \max(0, 0) = 0$ and $C_{1,1} = S_{1,1} + p_{1,1} = 0 + 3 = 3$. M_2 is the first machine in the no wait machine group $\mathbb{M}^1$. $C_{1,1} - C_{0,2} = 3 - 0 > d^1_{0,1,1} = 0$, $a = C_{1,1} - C_{0,2} - d^1_{0,1,1} = 3 - 0 - 0 = 3 > 0$. $C_{1,2} = C_{0,2} + d^1_{0,1,1} + p_{1,2} + a = 0 + 0 + 6 + 3 = 9$ and $C_{1,4} = C_{0,4} + d^2_{0,1,1} + a = 0 + 11 + 3 = 14$.
- J_2 is appended to π and π becomes $(J_0,\ J_1,\ J_2)$. $C_{2,1} = C_{1,1} + p_{2,1} = 3 + 5 = 8$. $a = \max\{0, 8 - 9 - 0\} = 0$ which implies that no right shift is needed for J_2 on the no-wait machine group $\mathbb{M}^1$. The completion time of J_2 on $\mathbb{M}^1_{[1]}$ (or M_2) is $C_{2,2} = C_{1,2} + d^1_{1,2,1} + p_{2,2} + a = 9 + 0 + 3 + 0 = 12$ and that on $\mathbb{M}^1_{[3]}$ (or M_4) is $C_{2,4} = C_{1,4} + d^2_{1,2,1} + a = 14 + 3 + 0 = 17$.
- After appending J_3 to π, $C_{3,1} = C_{2,1} + p_{3,1} = 8 + 6 = 14$. $C_{3,1} - C_{2,2} = 14 - 12 = 2 > d^1_{2,3,1} = 1$ and $a = C_{3,1} - C_{2,2} - d^1_{2,3,1} = 2 - 1 = 1$. $C_{3,2} = C_{2,2} + d^1_{2,3,1} + p_{3,2} + a = 12 + 1 + 2 + 1 = 16$ and $C_{3,4} = C_{2,4} + d^2_{2,3,1} + a = 17 + 5 + 1 = 23$. Therefore, the makespan of the final sequence is $C_{\max}(\pi) = 23$.

In addition, insertion and swap are two commonly used operators in many algorithm components such as initialization, reconstruction and local search. Similarly to the speed-up makespan calculation method for PFSP problems by Li et al. [11], we accelerate the makespan calculation for the pseudo-PFSP problem. Basically, there are two parts in each neighbor of a given solution: the unchanged part (the job subsequence is identical to the given solution) and changed part. We only need to calculate job completion times of changed parts to get makespans of neighbour solutions. Insertion and swap operators with this speed-up calculating method are called speed-up insertion and speed-up swap respectively. Though the worst case computational time complexities of the two operators cannot be improved by the speed-up calculating method, computation times of search processes with such operators can be greatly improved. For example, we carry out a single iteration of MIG (the proposed algorithm in this paper, to be detailed in the next section) with different speed-up operators on a random instance with 500 jobs to be scheduled on 20 machines. Without speed-up insertion and swap, computation times of MIG are 387.447 seconds and 267.331 seconds when we use GMC and SMC (excluding the computation time of matrix D) respectively. However, computation times of MIG with speed-up insertion and swap are 353.343 seconds and 211.905 seconds respectively. About 175 seconds are saved if all of the above speed-up methods are adopted and they are used in the proposed MIG algorithm. In relative terms, we are saving more than 88% of CPU time.

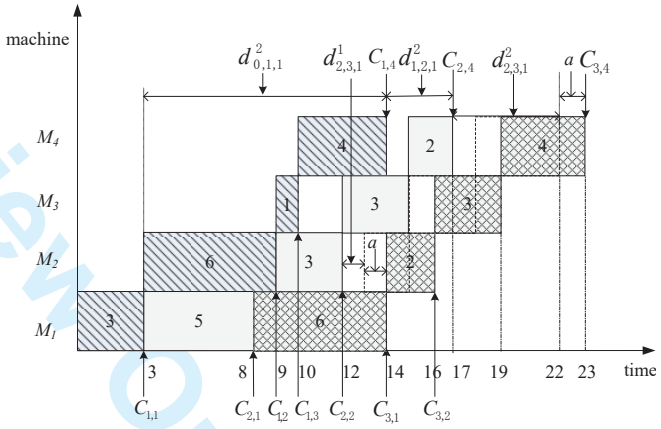

Fig. 4. Example of makespan calculation by SMC

V. Modified Iterated Greedy Method for MWFSP

The iterated greedy (IG) algorithm is now commonly adopted in flowshop scheduling problems [14], [20], [28]–[33] after it was first proposed by Ruiz and Stützle [29]. Basically, IG contains two operators: destruction and reconstruction. They are iteratively performed with a local search being applied optionally after reconstruction. Only a few parameters are needed which makes the method simple. Effectiveness and simplicity are the motivation to use the IG framework for the problem under study in this paper which contains the following components: Initialization, Destruction and Reconstruction, Local search and Acceptance criterion. Details of the components are described below.

A. Initialization

NEH [17] seems to be the best heuristic for flowshops, even better than some modern heuristics [30], [34]–[38]. Laha and Sarin [39] proposed a NEH-based constructive heuristic

for permutation flowshops. The main idea is simple: A job from the seed is inserted into the best position x of the current partial sequence by NEH and a new subsequence π is obtained. All jobs in π are adjusted by reinserting each job into all possible backward slots. FRB4$_k$ [40] is a trade-off method between NEH and the Dipak heuristic (proposed by Laha and Sarin [39]), in which the idea is that jobs far away from x are less likely to be affected. After a job from the seed has been inserted into the best position x by NEH, both the k jobs in the front of x and the same number in the back of x (i.e., $2k$ jobs in total) are adjusted by reinserting each job into all possible backward slots. For effectiveness and efficiency consideration, we adopt FRB4$_k$ in this paper with the above speed-up calculation methods as initialization. Details are shown in Algorithm 3. The time complexity of steps 2-5 is $O(mn)$, and that of step 6 is $O(nlogn)$. With the speed-up calculation methods, the time complexity of steps 8-11 is $O(kn^3(\xi+q))$. Therefore, the time complexity of Algorithm 3 is $O(kn^3(\xi+q))$.

ALGORITHM 3: MNEH(k) /*Initialization*/

```
1  begin
2      for j = 1 to n do
3          P_j = 0;
4          for i = 1 to m do
5              P_j = P_j + p_{i,j};
6      An initial sequence λ = {λ_[1], λ_[2], ..., λ_[n]} is
       obtained by sorting jobs in decreasing order of P_j;
7      π = (J_0, λ_[1]);
8      for i = 2 to n do
9          π ← Insert job λ_[i] into position x of π resulting
           in the minimum makespan;
10         for x' = max(1, x − k) to min(x + k, i) do
11             π ← Insert job π_[x'] into the position of π
               with the minimum makespan;
12     return π.
```

B. Modified Destruction and Reconstruction

In this paper, we propose a modified destruction and reconstruction (MDR for short) component to improve the diversification and intensification of the whole search process. The process is detailed in Algorithm 4. Both DR (the traditional destruction and reconstruction) and MDR adopt the same random destruction operation but utilize different reconstruction operations. Similarly to the above initialization component, the reconstruction of MDR reinserts the removed jobs using the FRB4$_k$ insertion [40]. In other words, MDR explores search spaces with wider ranges.

ALGORITHM 4: MDR(π, r, k) /*Modified Destruction and Reconstruction*/

```
1  begin
       // Destruction
2      B = ∅;
3      for i = 0 to r − 1 do
4          Randomly remove job B_i from π ;
5          B ← B ∪ {B_i};
       // Reconstruction
6      for i = 0 to r − 1 do
7          π ← Reinsert job B_[i] into position x of π
           resulting in best C_max;
8          for x' = max(1, x − k) to min(x + k, i) do
9              π ← Insert job B_[x'] into the position of π
               with the minimum makespan;
10     return π.
```

C. Local Search Methods

To enhance the intensification of the proposed MIG algorithm, local search methods are applied. Insertion neighborhood [29], [30], [41], [42], swap neighborhood and variable neighborhood searches (VNS) are commonly used as local search methods for scheduling problems, particularly flowshops. In addition, variable neighborhood descent (VND) [43] is a variant of the variable neighborhood search (VNS) [44]. In this paper, VND is adopted as the local search method containing $k_{\max}$ neighborhood structures as depicted in Algorithm 5. All neighborhood structures are explored sequentially and the i^{th} ($i = 1, \ldots, k_{\max}$) neighborhood structure induces neighborhood N_i VND starts from an initial solution π^*. A local minimum π^t is obtained after exploring the first neighborhood structure. If π^t is better than π^*, π^* is updated with π^t and the first neighborhood search is conducted again on π^*. Otherwise, the next neighborhood structure is examined on π^*. If a better solution is found after exploring a neighborhood, π^* is replaced and the search continues with the first neighborhood structure. VND stops only when no better solution can be found after all neighborhood structures are explored. VND can enhance the intensification of the search process for no-wait flowshop scheduling problems [1], [20].

Let π be the current solution, π^* be the current best solution, π^s be the seed and π^t be the newly constructed solution. The VND adopts the following first four existing neighborhood structures and the other two proposed ones.

1) CINS (Complete Insert Neighborhood Structure) initializes π^* and π^s to π. For each job $\pi^s_{[i]}$ ($i = 1, \ldots, n$), the following process is conducted: (i) π is assigned to π^t. (ii) $\pi^s_{[i]}$ is removed from π^t and tried to be inserted into all possible positions of the remaining sub-sequence $\pi^{'}$. (iii) π^t is updated with the best of the n newly constructed solutions. (iv) If π^t is better than π^*, π^* is replaced by π^t. The process is repeated until all n jobs in π^s are tried and π^* is the best found solution.
2) GINS (Greedy Insert Neighborhood Structure) was proposed by Ruiz et.al [14], [29]. π^* is initialized as π. π^s is randomly generated. For each job $\pi^s_{[i]}$ ($i = 1, \ldots, n$), the following process is conducted: (i) π is set as π^*. (ii)

ALGORITHM 5: VND(π)

```
begin
    π* = π; Flag = true;
    while (Flag = true) do
        Flag = false;
        i = 1;
        while i ≤ k_max do
            Find the best solution π^t in neighborhood N_k(π*);
            if C_max(π^t) < C_max(π*) then
                π* = π^t;
                i = 1;
            else
                i = i + 1;
        if C_max(π^t) < C_max(π*) then
            Flag = false;
    return π*.
```

$\pi^s_{[i]}$ is removed from π and tried to be inserted into all possible positions of the remaining sub-sequence $\pi^{'}$. (iii) π^t is updated with the best of the n newly constructed solutions. (iv) If π^t is better than π^*, π^* is replaced by π^t. The process is repeated until all n jobs in π^s are tried and π^* is the best found solution. The differences between GINS and CINS lie in two aspects: (a) π^s is randomly generated in GINS while it is assigned as π in CINS. (b) π is updated with a better solution found in an iteration in GINS while π remains unchanged in CINS.

3) CPINS (Complete Pair Insert Neighborhood Structure) is similar to CINS. The only difference is that CPINS removes and reinserts a pair of adjacent jobs rather than a single job each time. This neighborhood structure was adopted by Ding et.al [20].
4) CSNS (Complete Swap Neighborhood Structure) swaps each pair of jobs in π and returns the best found solution.
5) GPINS (Greedy Pair Insert Neighborhood Structure) is inspired from CPINS and GINS. For each job $\pi^s_{[i]}$ ($i = 1, \ldots, n$), the adjacent job pair starting from $\pi^s_{[i]}$ is removed from π and tried to be inserted into all possible positions of the remaining sub-sequence $\pi^{'}$. The other operations are identical to those of GINS.
6) CPSNS (Complete Pair Swap Neighborhood Structure) swaps each pair of two adjacent jobs in π and returns the best solution.

Time complexities of the six neighborhood structures are $O(n^2)$. According to the VND framework and the above neighborhood structures, seven VND algorithms are combined as shown in Table II.

TABLE II
LOCAL SEARCH METHODS BASED ON DIFFERENT NEIGHBORHOOD STRUCTURES

Local Search	Neighborhood Structures			
	N_1	N_2	N_3	N_4
VND_0	CINS	–	–	–
VND_1	GINS	–	–	–
VND_2	CSNS	CINS	–	–
VND_3	CINS	CSNS	–	–
VND_4	CSNS	CINS	CPINS	–
VND_5	CSNS	GINS	GPINS	CPSNS
VND_6	CSNS	GINS	CPSNS	GPINS

D. Acceptance Criterion

After performing the above operators on an initial solution π, a new solution π'' is obtained. If π'' is better than the incumbent π and best solution π^*, both π^* and π are replaced by π''. If π'' is not better than π^* but better than π, π is replaced by π''. Otherwise π is replaced by π'' with a certain probability. In this paper, we adopt the RPD-based probability determination using formula $e^{-\frac{C_{\max}(\pi'')-C_{\max}(\pi)}{C_{\max}(\pi)}\times 100}$ recently proposed by Hatami et al. [45] which is different from that introduced in the original IG of [29]. In other words, π is replaced by π'' with probability $e^{-\frac{C_{\max}(\pi'')-C_{\max}(\pi)}{C_{\max}(\pi)}\times 100}$ if π'' is not better than π. This acceptance criterion does not need the typical temperature parameter of IG methods, is simpler, and as shown in better performing.

E. Modified Iterated Greedy Algorithm

The proposed MIG (Modified Iterated Greedy) algorithm for MWFSPs is illustrated in Algorithm 6. An initial solution is constructed by MNEH. MDR and a selected local search are performed iteratively. Unlike the traditional DR, the number of removed jobs in MDR changes in each iteration of MIG. MDR starts from an initial size r_0, i.e., r_0 jobs are randomly removed from the current sequence π and they are reinserted into the remaining sub-sequence using the NEH insertion. If the new obtained solution $\pi^{''}$ after the local search is better than π, r is reset to r_0. Otherwise, r is unchanged. If no better solution is generated after 10 consecutive iterations (it was observed in preliminary experiments that there was no improvement after 10 iterations), r is increased by 1. The increase of r is limited to Δr, i.e., the number of removed jobs in the MDR is in the range $[r_0, r_0 + \Delta r]$.

F. Example

Let us consider an application from canned fruit processing industry. There are ten main phases in this manufacturing process: (1) Classification and pruning of fruits on an automatic sorter machine M_1. (2) Washing on Fruit on a washing machine M_2. (3) Removing peel and shell on Automatic nuclear removing machine (also called shelling machine) M_3. (4) Precooking on precooking machine M_4. (5) Packing into jars and adding sugar on canning equipment M_5. (6) Gas exhausting on exhaust equipment M_6. (7) Sealing on sealing machine M_7. (8) Sterilizing and refrigerating on machine M_8. (9) Labeling on machine M_9. (10) Packing on machine M_{10}. $M_4 \sim M_8$ are no-wait machines. The rest machines are regular machines. Let us assume that 15 fruit canning jobs (lots) $J_1, J_2, \cdots J_{15}$ are sequentially processed on these 10

ALGORITHM 6: Modified Iterated Greedy (MIG)

Input: Parameters r_0, Δr, k;
begin
  $r = r_0$, $x = 0$; // x is an iteration counter
  π is constructed by MNEH(k) and improved by a local search method LS; // one of the seven local search methods $\mathrm{VND}_0 \sim \mathrm{VND}_6$.
  $\pi^* = \pi$;
  while *(Termination criterion not satisfied)* **do**
    $\pi' \leftarrow$ MDR (π, r, k);
    $\pi'' \leftarrow$ Improved π' by LS;
    if $C_{\max}(\pi'') < C_{\max}(\pi)$ **then**
      $\pi = \pi''$; $x = 0$; $r = r_0$;
      if $C_{\max}(\pi'') < C_{\max}(\pi^*)$ **then**
        $\pi^* = \pi''$;
    else
      Randomly generate a number α in $[0, 1]$;
      if $\alpha \leq e^{-((C_{\max}(\pi'') - C_{\max}(\pi))/C_{\max}(\pi)) \times 100}$ **then**
        $\pi = \pi''$;
      $x = x + 1$;
      if $x > 10$ **then**
        $r = \min\{r + 1,\ r_0 + \Delta r\}$;
  return $C_{\max}(\pi^*)$.

machines. The corresponding processing times are shown in Table III. Figure 5 depicts the main process of the proposed MIG for the example. Firstly, the initial solution π is generated by MNEH and the makespan of π is 6120. Secondly, π is improved by VND_5 with makespan being 6087. The best found solution π^* is set as π. Thirdly, four jobs (J_{13}, J_{12}, J_1 and J_6) are removed from π and reinserted into positions of the remaining sub-sequence. The best sequence π' with makespan 6176 is obtained. By performing VND_5 on π', π'' is not improved, with a makespan of 6176. Because π'' is not better than π, π is replaced by π'' with a certain probability. Let us assume that the random number α is 0.21. Therefore, $e^{-((C_{\max}(\pi'') - C_{\max}(\pi))/C_{\max}(\pi)) \times 100}$ is 0.232 which is greater than α. As a result, π is replaced by π''. The destruction, reconstruction and local search operations are performed again and a better solution with makespan 6075 is found. The process (destruction, reconstruction and local search) is iteratively performed until the termination criterion ($n \times m/2 \times t = 15 \times 10/2 \times 30 = 2250ms$) is met. The final best solution $\pi^* = (J_0, J_2, J_4, J_{11}, J_{15}, J_3, J_{13}, J_{14}, J_7, J_8, J_{12}, J_1, J_6, J_9, J_{10}, J_5)$ is obtained with a makespan of 6034.

VI. Experimental Results

Since this work constitutes the first attempt at solving the MWFSP, there are no specific existing algorithms for it proposed in the literature. The presented MIG algorithm is

TABLE III
Processing times of canned fruit on ten machines

	M_1	M_2	M_3	M_4	M_5	M_6	M_7	M_8	M_9	M_{10}
J_1	63	540	420	216	963	144	143	30	196	38
J_2	28	80	360	112	428	52	55	50	784	31
J_3	84	420	480	133	294	112	205	55	785	130
J_4	60	20	0	3	60	12	10	23	83	11
J_5	60	90	180	78	105	27	15	59	57	29
J_6	240	80	420	40	96	88	26	24	97	84
J_7	60	80	180	100	132	68	44	27	136	136
J_8	30	90	120	87	57	66	200	34	123	31
J_9	105	420	480	49	98	119	110	46	80	42
J_{10}	32	120	300	48	244	32	96	14	84	37
J_{11}	120	120	60	14	98	52	13	19	130	21
J_{12}	200	300	480	60	220	90	144	27	427	323
J_{13}	90	270	660	180	954	90	363	54	850	38
J_{14}	105	420	480	91	434	84	128	23	62	95
J_{15}	60	120	540	48	600	120	315	24	424	20

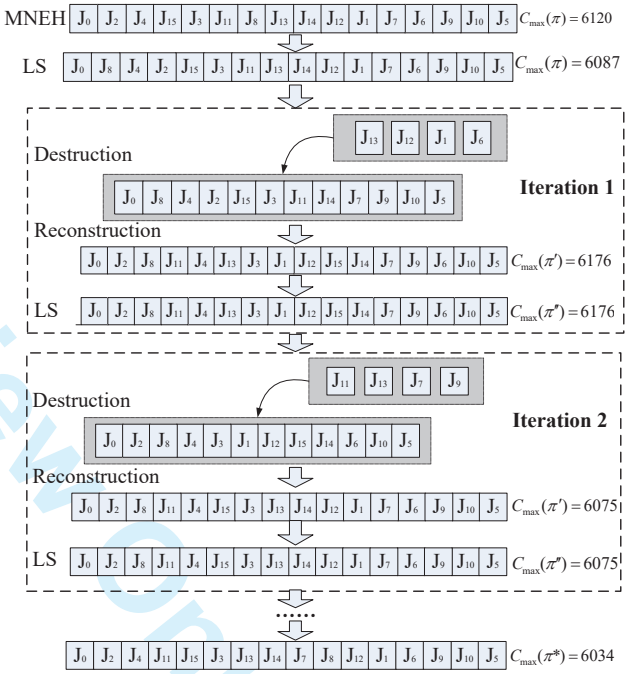

Fig. 5. Process of MIG for the fruit canning instance

evaluated on a large set of instances by comparing them with some existing algorithms for similar problems. We calibrate the parameters of the proposed algorithm first. The effectiveness of all algorithms is measured by the RPD (relative percentage deviation). Let S be the solution obtained by a given algorithm on a given instance and s_{best} be the best solution obtained in any experiment on the same instance. RPD is defined as

$$\mathrm{RPD} = \frac{s - s_{best}}{s_{best}} \times 100\% \tag{10}$$

All algorithms are coded in Visual C++ 2013 and run on computers with Windows XP professional, 1 GBytes RAM and Intel(R) Core(TM) i7-3770 processors running at 3.10 GHz. Experimental results are analyzed using the ANOVA (multi-factor analysis of variance) technique [46]. RPD is used as the response variable. The three main hypotheses (normali-

ty, homoscedasticity, and independence of the residuals) are checked. Since all the three hypotheses are close to zero, they are acceptable in this analysis.

A. Parameter Calibration

Besides the four mentioned parameters: r_0, Δr, k and the type of local search method VND_i $(i = 0, 1, \ldots, 6)$, the makespan calculation (denoted as C^m) has an influence on the response variable. The tested values of the five parameters to be calibrated are shown in Table IV. In total, there are $2 \times 7 \times 3 \times 3 \times 6 = 756$ combinations.Each combination is tested over a set of randomly generated MWFSP instances. Processing times are uniformly distributed in [1, 20]. The number of no-wait machines is uniformly distributed in [1, m] where m is the number of machines. Five instances are generated for a pair of job and machine sizes: 50×5, 50×10, 50×20, 100×5, 100×10, 100×20, 200×5, 200×10 and 200×20. Each combination is run five times on each instance. The termination criterion is $n \times (m/2) \times t$, where $t \in \{30, 60, 90\}$ milliseconds. Therefore, there are $9 \times 5 \times 5 \times 3 \times 756 = 510300$ experimental results.

TABLE IV
PARAMETERS TESTED IN THE CALIBRATION

Paramter	value
C^m	GMC, SMC
LS	VND_0, VND_1, VND_2, VND_3, VND_4, VND_5, VND_6
r_0	2, 4, 6
Δr	0, 2, 4
k	0, 2, 4, 6, 8, 10

After carrying out the statistical analyses, we show the means plot and 95.0% confidence level Tukey Honest Significant Difference (HSD) intervals in Figures 6∼10. Overlapping confidence intervals imply statistical insignificance in the corresponding averages, i.e., the observed differences in the overlapped means is not statistically significant at the indicated confidence level. From Figure 6, it can be observed that the proposed MIG with the SMC calculation is significantly better than that with the GMC computation even though matrix D is calculated in advance in SMC. It has to be noted that the same CPU time is allowed for all treatments with the same t value, therefore, the better results obtained by MIG with SMC calculation are due to the fact that SMC being faster, more iterations of MIG are possible and better solutions are obtained. Therefore, SMC is adopted in the following experiments.

Figure 7 shows the means plot and 95.0% confidence level Tukey HSD intervals for various local search methods on random instances. From Figure 7, we can observe that the differences are statistically significant for the different local search methods. RPD of the proposed algorithm is the least when local search methods are VND_5 and VND_6. The reason lies in that intensification of the proposal can be enhanced by the VNDs with CSNS, GINS, CPSNS, GPINS neighborhood structures. VND_2, VND_3, VND_4 perform better than VND_0 and VND_1 which indicates that a VND with more than one neighborhood structure usually performs better than one with single neighborhood structure. In addition, RPD of VND_1 is less than that of VND_0. Since VND_5 and VND_6 have similar performance, we adopt VND_5 as the local search method of the proposed MIG algorithm in the following experiments. It is observed in Figure 8 and Figure 9 that similarly for r_0 and Δr, the best RPD values occur when $r_0 = 4$ and $\Delta r = 2$. Therefore, we use $r_0 = 4$ and $\Delta r = 2$ in the following experiments. Figure 10 indicates that the differences are statistically significant for different k values. There is a clear tendency that the difference becomes smaller with the increase of k. But when $k \geq 6$ RPD has no further substantial improvement. Therefore, we set k to 8 for the proposed MIG algorithm in this paper.

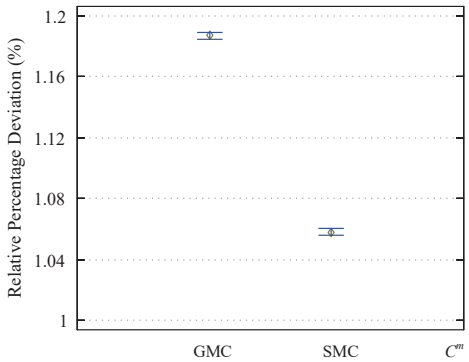

Fig. 6. Means plot and 95.0% confidence level Tukey HSD intervals for different makespan calculation methods on random instances

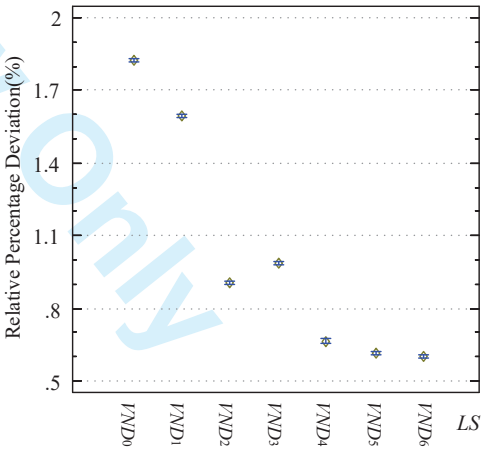

Fig. 7. Means plot and 95.0% confidence level Tukey HSD intervals for various local search methods on random instances

B. Algorithm Comparisons

According to the calibrated parameters and components, the proposed calibrated MIG algorithm is compared with some algorithms for similar problems. As stated, there are no existing methods for the MWFSP, so we modify five classical state-of-the-art algorithms for permutation flowshops and no-wait flowshop problems from the literature. Note that these are the closest possible competitors: TMIIG (improved iterated greedy algorithm with a Tabu-based reconstruction strategy) by Ding et al. [20], the original IG (iterated greedy) by Ruiz

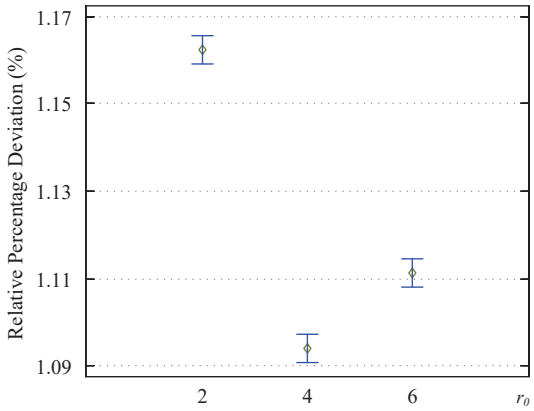

Fig. 8. Means plot and 95.0% confidence level Tukey HSD intervals for parameter r_0 on random instances

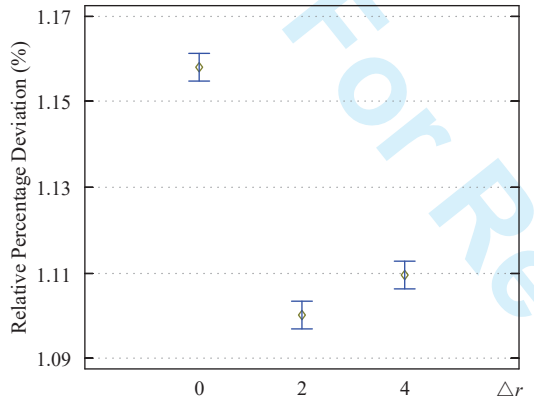

Fig. 9. Means plot and 95.0% confidence level Tukey HSD intervals for parameter Δr on random instances

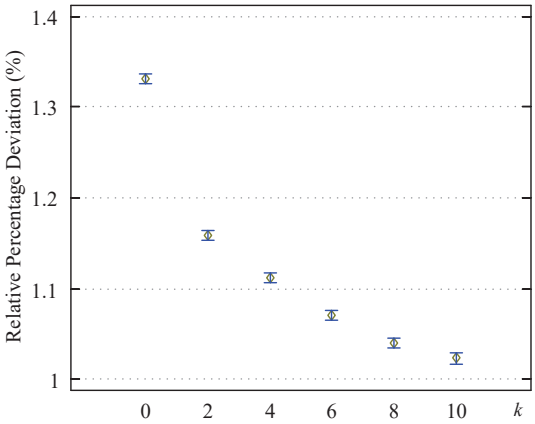

Fig. 10. Means plot and 95.0% confidence level Tukey HSD intervals for parameter k on random instances

et al. [29], IIGA (improved iterated greedy algorithm) by Pan and Wang [14], HDPSO (hybrid discrete particle swarm optimization) by Pan and Wang [24] and DPSO_{VND} (discrete particle swarm optimization algorithm with VND local search) by Pan et al. [1]. All the algorithms have been adapted for the MWFSP.

Since there are no MWFSP benchmark instances, we built a new benchmark following the benchmark proposed by Pan and Ruiz [30]. Note that this benchmark is different from the calibration instances so that to avoid overfitting in the calibrated proposed MIG. Based on Taillard benchmark [47], we generate 12 subsets of instances with $n \in \{20, 50, 100, 200, 500\}$ and $m \in \{5, 10, 20\}$. Each subset consists of ten instances. There are 120 instances in total. Similarly to Pan and Ruiz [30], seven different families of mixed no-wait flowshop problems are generated, i.e., there are 840 instances in total for the following algorithm comparisons.

- Family 1: The first 50% of the machines have the no-wait constraint. The remaining 50% machines can wait.
- Family 2: The second 50% of the machines have the no-wait constraint.
- Family 3: The machines alternate, in order, between regular and no-wait constraints.
- Family 4: A random 25% of the machines have the no-wait constraint.
- Family 5: A random 50% of the machines are no-wait.
- Family 6: 75% random no-wait machines.
- Family 7: All machines are no-wait.

Note that instances in the last family are pure no-wait problems, over which the proposed MIG algorithm is compared against existing methods which were also specifically proposed for full no-wait problems. The generated instances and results are available on the website http://webplus.seu.edu.cn/lxp/2016/1216/c12114a180375/page.psp. All algorithms adopt the speed-up makespan calculation (SMC) and the speed-up neighborhood search presented in this paper for a fair and accurate comparison. The termination criterion is set as $n \times (m/2) \times t$ milliseconds where $t \in \{30, 60, 90\}$. Every algorithm is run for five replications on each instance. Therefore, $840 \times 3 \times 6 \times 5 = 75600$ tests are conducted. This is very large set of results for which no additional replicates are needed and powerful statistical analyses can be conducted.

Figure 11 shows the interactions between the tested algorithms and the termination criterion $t \in \{30, 60, 90\}$ with 95.0% confidence level Tukey HSD intervals. Figure 11 illustrates that the proposed MIG algorithm is statistically better than TMIIG, DPSO_{VND}, HDPSO, IG and IIG for all the three t cases. In addition, in most cases results are not statistically better with higher t values, meaning that $t = 30$ is, on average, already enough time for most algorithms to converge. Therefore, we only show results of the compared algorithms with $t = 30$ in the following.

Table V shows the ARPDs (Average Relative Percentage Deviation) of the six algorithms grouped by $n \times m$ and family. The ARPD of each algorithm in every n, m and family combination is the average RPD on 10×5=50 instances. Table V shows that the average ARPD of MIG in all instances is 0.38, which is less than those of TMIIG, DPSO_{VND}, HDPSO, IG and IIG with 0.68, 0.66, 1.03, 1.25 and 1.12 respectively, i.e., MIG is the best algorithm among the tested algorithms for the MWFSPs. In addition, MIG obtains the 79 best results out of $12 \times 7 = 84$ cases when $t = 30$. ARPD differences between MIG and the other algorithms increase with the value of n. For example, the ARPD of MIG in group 20×5 and family 5 is 0.00 while those of the other algorithms are 0.01, 0.01, 0.13, 0.02 and 0.11 respectively. However, the ARPD

of MIG in group 500×20 of family 5 is 0.50 and those of the other algorithms are 3.08, 1.31, 2.06, 1.57 and 1.74 respectively. Figure 12 and Figure 13 show the interactions between the compared algorithms and m (n) with $t = 30$ with 95.0% confidence level Tukey HSD intervals. From Figure 12 and Figure 13, we can observe that MIG is much more robust than the other compared algorithms, i.e., the problem size has little influence on the performance of MIG whereas it exerts great influence on the the performance of the other five. In addition, MIG always gets the smallest RPD among the compared algorithms for any problem size. Bigger problem size demonstrates better performance of MIG. Figure 14 shows the interactions between the compared algorithms and *Families* with $t = 30$ with 95.0% confidence level Tukey HSD intervals. From Figure 14, we can observe that problem structures exert little influence on the performance of MIG. However, parameter *Family* has different impacts on performance of the other five algorithms. MIG always obtains the smallest RPD and statistically outperforms the other algorithms, even for pure no-wait problems (*Family* 7).

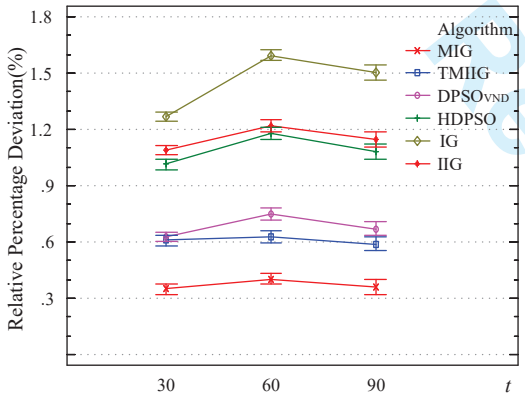

Fig. 11. Interactions between the tested algorithms and the termination criterion $t \in \{30, 60, 90\}$ with 95.0% confidence level Tukey HSD intervals.

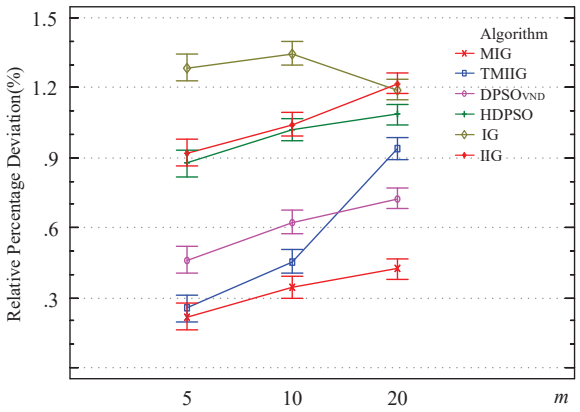

Fig. 12. Interactions between the compared algorithms and m with $t = 30$ and 95.0% confidence level Tukey HSD intervals.

VII. Conclusions and Future Work

In this paper, we have studied mixed no-wait flowshop problems (MWFSP), to the best of our knowledge, for the first

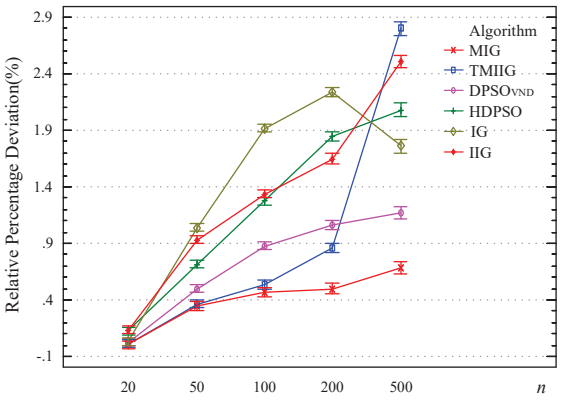

Fig. 13. Interactions between the compared algorithms and n with $t = 30$ and 95.0% confidence level Tukey HSD intervals.

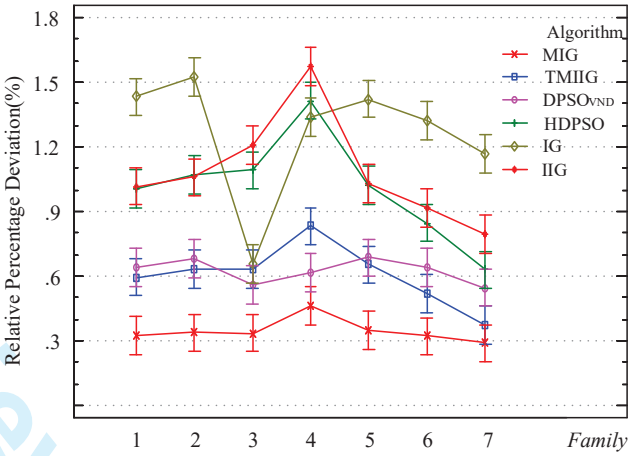

Fig. 14. Interactions between the compared algorithms and Families with $t = 30$ and 95.0% confidence level Tukey HSD intervals.

time. This kind of problem is more realistic in practice and it is a generalization of both the traditional permutation flowshop and the no-wait flowshop problems. Based on the established mathematical model and characteristics of the MWFSP, we designed two makespan calculation methods, GMC and SMC. Generally, SMC is faster than GMC though their worst computational time complexities are the same. A modified iterated greedy (MIG) algorithm is proposed for MWFSPs. Starting with an initial solution generated by MNEH, three phases are run iteratively: destruction, reconstruction and local search. To enhance the diversification of MIG (i.e., to avoid trapping into local optimum), the number of removed jobs in the destruction phase changes dynamically during the search process of MIG. To improve the intensification and efficiency of MIG, the idea of $FRB4_k$ was adopted in MNEH and MDR phases. Two effective VND local searches were developed which were based on four neighborhood structures CSNS, GINS, GPINS, CPSNS. MIGs with the two VNDs show better performance than existing algorithms. All parameters with different levels and components with various candidates were calibrated by ANOVA. After comparing the proposed MIG algorithm with five existing state-of-the-art algorithms for similar problems in a large set of instances, experimental results showed that MIG outperforms TMIIG, HDPOS, IG, IIG, and $DPSO_{VND}$.

TABLE V
AVERAGE RELATIVE PERCENTAGE DEVIATION FOR ALL COMPARED ALGORITHMS WITH t=30.

$n \times m$	Family	TMIIG	$DPSO_{VND}$	HDPSO	IG	IIG	MIG	$n \times m$	Family	TMIIG	$DPSO_{VND}$	HDPSO	IG	IIG	MIG
20×5	1	0.03	**0.01**	0.22	0.05	0.21	0.02	100×5	1	0.53	0.91	2.09	2.78	1.80	**0.31**
20×5	2	0.02	0.01	0.08	0.06	0.14	**0.00**	100×5	2	0.41	0.91	1.80	2.64	1.66	**0.33**
20×5	3	0.10	0.15	0.18	0.14	0.39	**0.02**	100×5	3	0.21	0.29	0.74	0.80	0.65	**0.12**
20×5	4	**0.02**	**0.02**	0.22	0.04	0.23	0.03	100×5	4	0.54	0.90	2.10	2.79	1.94	**0.36**
20×5	5	0.01	0.01	0.13	0.02	0.11	**0.00**	100×5	5	**0.46**	1.16	1.92	3.02	1.84	0.53
20×5	6	0.02	0.01	0.12	0.01	0.09	**0.00**	100×5	6	0.57	1.16	1.97	3.02	1.95	**0.51**
20×5	7	0.01	0.02	0.11	0.03	0.18	**0.00**	100×5	7	0.48	0.98	1.42	2.50	1.45	**0.45**
Average		0.03	0.03	0.15	0.05	0.19	**0.01**	Average		0.46	0.90	1.72	2.51	1.61	**0.37**
20×10	1	**0.00**	0.02	0.08	**0.00**	0.05	**0.00**	100×10	1	0.71	1.05	1.18	2.33	1.38	**0.64**
20×10	2	**0.00**	0.01	0.30	0.03	0.14	**0.00**	100×10	2	0.62	1.16	1.54	2.63	1.49	**0.56**
20×10	3	0.01	0.02	0.09	0.02	0.19	**0.00**	100×10	3	0.48	0.64	1.18	0.88	1.24	**0.26**
20×10	4	0.03	0.07	0.15	0.04	0.26	**0.01**	100×10	4	0.76	0.87	1.56	1.55	1.54	**0.48**
20×10	5	**0.00**	0.02	0.08	0.01	0.05	**0.00**	100×10	5	**0.53**	0.98	1.21	2.28	1.30	0.56
20×10	6	**0.00**	0.03	0.05	0.00	0.07	**0.00**	100×10	6	**0.48**	0.90	0.76	1.90	1.06	0.55
20×10	7	0.01	0.01	0.15	0.07	0.11	**0.00**	100×10	7	**0.41**	0.77	0.66	1.60	0.92	0.47
Average		0.01	0.03	0.13	0.02	0.13	**0.00**	Average		0.57	0.91	1.16	1.88	1.27	**0.50**
20×20	1	0.01	0.02	0.21	0.04	0.15	**0.00**	100×20	1	**0.45**	0.82	0.80	1.77	1.03	0.58
20×20	2	0.01	0.01	0.04	0.01	0.03	**0.00**	100×20	2	**0.51**	0.79	0.66	1.75	0.95	0.56
20×20	3	0.02	0.06	0.10	**0.01**	0.11	0.02	100×20	3	0.76	0.83	1.57	0.81	1.69	**0.55**
20×20	4	0.01	0.04	0.07	0.01	0.10	**0.01**	100×20	4	0.83	0.99	1.65	1.10	1.61	**0.49**
20×20	5	0.00	0.00	0.09	0.00	0.07	**0.00**	100×20	5	0.63	0.95	0.81	1.51	0.99	**0.57**
20×20	6	0.02	**0.01**	0.14	0.02	0.07	0.02	100×20	6	0.51	0.82	0.73	1.35	0.90	**0.49**
20×20	7	0.01	**0.00**	0.05	0.08	0.02	**0.00**	100×20	7	0.38	0.56	0.47	1.18	0.71	**0.46**
Average		**0.01**	0.02	0.10	0.02	0.08	**0.01**	Average		0.58	0.82	0.96	1.35	1.13	**0.53**
50×5	1	0.23	0.45	0.88	1.37	1.06	**0.22**	200×10	1	0.86	1.01	2.15	2.80	1.76	**0.50**
50×5	2	**0.23**	0.38	0.73	1.26	1.03	0.25	200×10	2	1.00	1.30	2.76	3.15	2.17	**0.61**
50×5	3	0.09	0.10	0.23	0.30	0.26	**0.05**	200×10	3	0.76	0.91	2.23	1.15	1.78	**0.30**
50×5	4	**0.18**	0.41	0.88	1.40	1.05	0.26	200×10	4	0.85	0.81	2.38	2.13	1.88	**0.40**
50×5	5	0.43	0.65	0.94	1.76	1.18	**0.39**	200×10	5	0.95	1.11	2.18	2.77	1.78	**0.45**
50×5	6	0.40	0.63	0.97	1.63	1.20	**0.38**	200×10	6	0.92	1.12	1.79	2.61	1.63	**0.46**
50×5	7	0.39	0.55	0.70	1.34	0.94	**0.39**	200×10	7	0.54	0.99	1.11	2.15	1.47	**0.46**
Average		**0.28**	0.45	0.76	1.29	0.96	**0.28**	Average		0.84	1.04	2.08	2.39	1.78	**0.46**
50×10	1	0.40	0.56	0.56	1.02	0.82	**0.37**	200×20	1	0.72	1.12	1.57	2.39	1.47	**0.66**
50×10	2	**0.32**	0.48	0.69	1.33	0.79	**0.42**	200×20	2	0.97	1.20	1.61	2.49	1.68	**0.64**
50×10	3	0.62	0.81	1.25	0.90	1.75	**0.55**	200×20	3	1.22	1.12	2.34	1.05	1.90	**0.55**
50×10	4	**0.50**	0.59	1.01	0.87	1.46	0.57	200×20	4	1.28	1.08	2.52	2.27	1.72	**0.46**
50×10	5	0.40	0.48	0.64	1.13	0.92	**0.34**	200×20	5	0.91	1.11	1.64	2.11	1.52	**0.48**
50×10	6	**0.30**	0.40	0.42	0.92	0.60	0.34	200×20	6	0.68	1.04	1.01	1.86	1.20	**0.52**
50×10	7	0.29	0.33	0.42	0.81	0.56	**0.28**	200×20	7	0.37	1.01	0.61	1.80	1.13	**0.54**
Average		**0.40**	0.52	0.71	1.00	0.99	0.41	Average		0.88	1.10	1.61	2.00	1.52	**0.55**
50×20	1	0.35	0.46	0.50	0.76	0.58	**0.29**	500×20	1	2.87	1.27	1.82	1.65	1.76	**0.31**
50×20	2	0.48	0.49	0.56	0.90	0.72	**0.32**	500×20	2	3.04	1.44	2.09	1.96	1.95	**0.36**
50×20	3	**0.51**	0.71	1.03	0.71	1.53	0.55	500×20	3	2.84	1.07	2.17	0.91	3.07	**1.05**
50×20	4	0.60	0.78	1.07	0.88	1.29	**0.51**	500×20	4	4.39	0.86	3.35	2.88	5.51	**1.94**
50×20	5	0.43	0.48	0.58	0.87	0.72	**0.34**	500×20	5	3.08	1.31	2.06	1.57	1.74	**0.50**
50×20	6	0.31	0.38	0.57	0.93	0.62	**0.23**	500×20	6	2.05	1.19	1.64	1.60	1.58	**0.38**
50×20	7	0.27	0.31	0.40	0.69	0.53	**0.19**	500×20	7	1.34	1.05	1.44	1.65	1.57	**0.24**
Average		0.42	0.52	0.67	0.82	0.86	**0.35**	Average		2.80	1.17	2.08	1.75	2.46	**0.68**
								Global average		0.68	0.66	1.03	1.25	1.12	**0.38**

MWFSPs with other objectives (e.g., total flowtime) are common in practical industries and are promising topics to investigate in the future. Further refinements of the algorithm procedures are also interesting future avenues of research.

ACKNOWLEDGMENT

This work is supported by the National Natural Science Foundation of China (Grants 61572127, 61272377) and the Key Research & Development program in Jiangsu Province (No. BE2015728). Rubén Ruiz is supported by the Spanish Ministry of Economy and Competitiveness, under the project "SCHEYARD-Optimization of scheduling problems in container yards" (No. DPI2015-65895-R) partly financed with FEDER funds.

REFERENCES

[1] Q.-K. Pan, M. F. Tasgetiren, and Y.-C. Liang, "A discrete particle swarm optimization algorithm for the no-wait flowshop scheduling problem," *Computers & Operations Research*, vol. 35, no. 9, pp. 2807–2839, 2008.

[2] J.-q. Li, Q.-k. Pan, and P.-y. Duan, "An improved artificial bee colony algorithm for solving hybrid flexible flowshop with dynamic operation skipping," *IEEE transactions on cybernetics*, vol. 46, no. 6, pp. 1311–1324, 2016.

[3] S. Nguyen, M. Zhang, M. Johnston, and K. C. Tan, "Automatic programming via iterated local search for dynamic job shop scheduling." *IEEE Transactions on Cybernetics*, vol. 45, no. 1, pp. 1 – 14, 2015.

[4] J.-Q. Li, Q.-K. Pan, and K. Mao, "A hybrid fruit fly optimization algorithm for the realistic hybrid flowshop rescheduling problem in steelmaking systems," *IEEE Transactions on Automation Science and Engineering*, vol. 13, no. 2, pp. 932–949, 2016.

[5] H. Yuan, J. Bi, W. Tan, M. Zhou, B. H. Li, and J. Li, "Ttsa: An effective scheduling approach for delay bounded tasks in hybrid clouds." *IEEE Transactions on Cybernetics*, 2016.

[6] L. Gao, G. Zhang, L. Zhang, and X. Li, "An efficient memetic algorithm for solving the job shop scheduling problem," *Computers & Industrial Engineering*, vol. 60, no. 4, pp. 699–705, 2011.
[7] R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. R. Kan, "Optimization and approximation in deterministic sequencing and scheduling: a survey," *Annals of Discrete Mathematics*, vol. 5, pp. 287–326, 1979.
[8] A. H. G. RinnooyKan, "Machine scheduling problem: Classification," *Complexity and Computation, Nijhoff, The Hague*, 1976.
[9] C. H. Papadimitriou and P. C. Kanellakis, "Flowshop scheduling with limited temporary storage," *Journal of the ACM (JACM)*, vol. 27, no. 3, pp. 533–549, 1980.
[10] P. J. Kalczynski and J. Kamburowski, "On no-wait and no-idle flow shops with makespan criterion," *European Journal of Operational Research*, vol. 178, no. 3, pp. 677–685, 2007.
[11] X. Li, Q. Wang, and C. Wu, "Efficient composite heuristics for total flowtime minimization in permutation flow shops," *Omega*, vol. 37, no. 1, pp. 155–164, 2009.
[12] C. Wang, X. Li, and Q. Wang, "Accelerated tabu search for no-wait flowshop scheduling problem with maximum lateness criterion," *European Journal of Operational Research*, vol. 206, no. 1, pp. 64–72, 2010.
[13] X. Li, Q. Wang, and C. Wu, "Heuristic for no-wait flow shops with makespan minimization," *International Journal of Production Research*, vol. 46, no. 9, pp. 2519–2530, 2008.
[14] Q.-K. Pan, L. Wang, and B.-H. Zhao, "An improved iterated greedy algorithm for the no-wait flow shop scheduling problem with makespan criterion," *The International Journal of Advanced Manufacturing Technology*, vol. 38, no. 7-8, pp. 778–786, 2008.
[15] I. Adiri and D. Pohoryles, "Flowshop/no-idle or no-wait scheduling to minimize the sum of completion times," *Naval Research Logistics Quarterly*, vol. 29, no. 3, pp. 495–504, 1982.
[16] S. M. Johnson, "Optimal two-and three-stage production schedules with setup times included," *Naval Research Logistics Quarterly*, vol. 1, no. 1, pp. 61–68, 1954.
[17] M. Nawaz, E. E. Enscore, and I. Ham, "A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem," *Omega*, vol. 11, no. 1, pp. 91–95, 1983.
[18] R. Haupt, "A survey of priority rule-based scheduling," *Operations-Research-Spektrum*, vol. 11, no. 1, pp. 3–16, 1989.
[19] R. Ramasesh, "Dynamic job shop scheduling: a survey of simulation research," *Omega*, vol. 18, no. 1, pp. 43–57, 1990.
[20] J.-Y. Ding, S. Song, J. N. Gupta, R. Zhang, R. Chiong, and C. Wu, "An improved iterated greedy algorithm with a tabu-based reconstruction strategy for the no-wait flowshop scheduling problem," *Applied Soft Computing*, vol. 30, pp. 604–613, 2015.
[21] S. U. Sapkal and D. Laha, "A heuristic for no-wait flow shop scheduling," *The International Journal of Advanced Manufacturing Technology*, vol. 68, no. 5-8, pp. 1327–1338, 2013.
[22] E. Bertolissi, "Heuristic algorithm for scheduling in the no-wait flow-shop," *Journal of Materials Processing Technology*, vol. 107, no. 1, pp. 459–465, 2000.
[23] G. Liu, S. Song, and C. Wu, "Some heuristics for no-wait flowshops with total tardiness criterion," *Computers & Operations Research*, vol. 40, no. 2, pp. 521–525, 2013.
[24] Q.-K. Pan, L. Wang, M. F. Tasgetiren, and B.-H. Zhao, "A hybrid discrete particle swarm optimization algorithm for the no-wait flow shop scheduling problem with makespan criterion," *The International Journal of Advanced Manufacturing Technology*, vol. 38, no. 3-4, pp. 337–347, 2008.
[25] L.-Y. Tseng and Y.-T. Lin, "A hybrid genetic algorithm for no-wait flowshop scheduling problem," *International Journal of Production Economics*, vol. 128, no. 1, pp. 144–152, 2010.
[26] B. Jarboui, M. Eddaly, and P. Siarry, "A hybrid genetic algorithm for solving no-wait flowshop scheduling problems," *The International Journal of Advanced Manufacturing Technology*, vol. 54, no. 9-12, pp. 1129–1143, 2011.
[27] A. AitZai, B. Benmedjdoub, and M. Boudhar, "Branch-and-bound and PSO algorithms for no-wait job shop scheduling," *Journal of Intelligent Manufacturing*, vol. 27, pp. 679–688, 2016.
[28] J. Grabowski and J. Pempera, "Some local search algorithms for no-wait flow-shop problem with makespan criterion," *Computers & Operations Research*, vol. 32, no. 8, pp. 2197–2212, 2005.
[29] R. Ruiz and T. Stützle, "A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem," *European Journal of Operational Research*, vol. 177, no. 3, pp. 2033–2049, 2007.
[30] Q.-K. Pan and R. Ruiz, "An effective iterated greedy algorithm for the mixed no-idle permutation flowshop scheduling problem," *Omega*, vol. 44, pp. 41–50, 2014.
[31] S. W. Lin, K. C. Ying, W. J. Wu, and Y. I. Chiang, "Multi-objective unrelated parallel machine scheduling: a tabu-enhanced iterated pareto greedy algorithm," *International Journal of Production Research*, pp. 1–12, 2016.
[32] S. Lin, K. Ying, and C. Huang, "Minimising makespan in distributed permutation flowshops using a modified iterated greedy algorithm," *International Journal of Production Research*, vol. 51, no. 16, pp. 5029–5038, 2013.
[33] K. C. Ying, S. W. Lin, and S. Y. Wan, "Bi-objective reentrant hybrid flowshop scheduling: an iterated pareto greedy algorithm," *International Journal of Production Research*, vol. 52, no. 19, pp. 5735–5747, 2014.
[34] R. Ruiz and C. Maroto, "A comprehensive review and evaluation of permutation flowshop heuristics," *European Journal of Operational Research*, vol. 165, no. 2, pp. 479–494, 2005.
[35] Q.-K. Pan and R. Ruiz, "A comprehensive review and evaluation of permutation flowshop heuristics to minimize flowtime," *Computers & Operations Research*, vol. 40, no. 1, pp. 117–128, 2013.
[36] S. Song, "Accelerated methods for total tardiness minimisation in no-wait flowshops," *International Journal of Production Research*, vol. 53, no. 4, pp. 1002–1018, 2015.
[37] C. Wang, X. Li, and Q. Wang, "Accelerated tabu search for no-wait flowshop scheduling problem with maximum lateness criterion," *European Journal of Operational Research*, vol. 206, no. 1, pp. 64–72, 2010.
[38] J. Ding, S. Song, R. Zhang, J. N. Gupta, and C. Wu, "Accelerated methods for total tardiness minimisation in no-wait flowshops," *International Journal of Production Research*, vol. 53, no. 4, pp. 1002–1018, 2015.
[39] D. Laha and S. C. Sarin, "A heuristic to minimize total flow time in permutation flow shop," *Omega*, vol. 37, no. 3, pp. 734–739, 2009.
[40] S. F. Rad, R. Ruiz, and N. Boroojerdian, "New high performing heuristics for minimizing makespan in permutation flowshops," *Omega*, vol. 37, no. 2, pp. 331–345, 2009.
[41] Q.-K. Pan and R. Ruiz, "An estimation of distribution algorithm for lot-streaming flow shop problems with setup times," *Omega*, vol. 40, no. 2, pp. 166–180, 2012.
[42] E. Vallada and R. Ruiz, "Cooperative metaheuristics for the permutation flowshop scheduling problem," *European Journal of Operational Research*, vol. 193, no. 2, pp. 365–376, 2009.
[43] C. Blum and A. Roli, "Metaheuristics in combinatorial optimization: Overview and conceptual comparison," *ACM Computing Surveys (CSUR)*, vol. 35, no. 3, pp. 268–308, 2003.
[44] N. Mladenović and P. Hansen, "Variable neighborhood search," *Computers & Operations Research*, vol. 24, no. 11, pp. 1097–1100, 1997.
[45] S. Hatami, R. Ruiz, and C. Andrés-Romano, "Heuristics and metaheuristics for the distributed assembly permutation flowshop scheduling problem with sequence dependent setup times," *International Journal of Production Economics*, vol. 169, pp. 76–88, 2015.
[46] T. Bartz-Beielstein, M. Chiarandini, L. Paquete, and Preuss, Eds., *Experimental methods for the analysis of optimization algorithms, Springer*.
[47] E. Taillard, "Benchmarks for basic scheduling problems," *european journal of operational research*, vol. 64, no. 2, pp. 278–285, 1993.

Yamin Wang received her B.Sc. degree in Computer Science and Education from Liaocheng University, Shandong, China, in 2001. She obtained her M.Sc. degree in Computer Science and Technology from Beijing University of Technology, Beijing, China, in 2009. She is currently a Ph.D Candidate at the School of Computer Science and Engineering, Southeast University. She is the author or co-author of some academic papers, such as *IEEE Transaction on Systems, Man, and Cybernetics: Systems*, *IEEE International Conference on Systems, Man, and Cybernetics*, *Computers & Operations Research*. Her research interest focuses on algorithm design and scheduling optimization.

Xiaoping Li (M09-SM12) received his B.Sc. and M.Sc. degrees in Applied Computer Science from the Harbin University of Science and Technology, Harbin, China, in 1993 and 1999 respectively. He obtained his Ph.D. degree in Applied Computer Science from the Harbin Institute of Technology, Harbin, China, in 2002. He joined Southeast University, Nanjing, China, in 2005, and is currently a full professor at the School of Computer Science and Engineering. He is the author or co-author over more than 100 academic papers, some of which have been published in international journals such as *IEEE Transactions on Automation Science and Engineering*, *IEEE Transactions on Services Computing*, *IEEE Transaction on Systems, Man, and Cybernetics: Systems*, *Omega*, *European Journal of Operational Research*, *Information Sciences*. His research interests focus on Scheduling in Cloud Computing, Scheduling in Cloud Manufacturing, Machine Scheduling, Project Scheduling and Terminal Container Scheduling.

Rubén Ruiz is a full professor of Statistics and Operations Research at the Polytechnic University of Valencia, Spain. He is co-author of more than 50 papers in International Journals and has participated in presentations of more than a hundred papers at national and international conferences. He is editor of the Elseviers journal *Operations Research Perspectives (ORP)* and co-editor of the JCR-listed journal *European Journal of Industrial Engineering (EJIE)*. He is also associate editor of other important journals like *TOP* or *Applied Mathematics and Computation* as well as member of the editorial boards of several journals most notably *European Journal of Operational Research* and *Computers and Operations Research*. He is the director of the Applied Optimization Systems Group (SOA, http://soa.iti.es) at the Instituto Tecnolgico de Informtica (ITI, http://www.iti.es) where he has been principal investigator in several public research projects as well as privately funded projects with industrial companies. His research interests include scheduling and routing in real life scenarios.

Shaochun Sui received his B.Sc., M.Sc. and Ph.D. degrees in Mechanical Engineering from the Tsinghua University, Beijing, China, in 2007, 2009 and 2016 respectively. He joined Chengdu Aircraft Company, Chengdu, China, in 2009. Currently he is the Director of the Production Management of CAC. He is the author or co-author over more than 20 academic papers. His research interests focus on topics in advanced manufacturing systems.