Applied Soft Computing Journal 96 (2020) 106629

Contents lists available at ScienceDirect

Applied Soft Computing Journal

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

An effective Iterated Greedy algorithm for the distributed permutation flowshop scheduling with due windows

Xue-Lei Jing [a,b], Quan-Ke Pan [a,c,*], Liang Gao [d], Yu-Long Wang [a]

[a] School of Mechatronic Engineering and Automation, Shanghai University, Shanghai 200072, PR China
[b] Experimental and Network Information Center, Liaocheng University, Liaocheng 252000, PR China
[c] School of Computer Science and Technology, Liaocheng University, Liaocheng 252000, PR China
[d] State Key Lab of Digital Manufacturing Equipment & Technology in Huazhong University of Science & Technology, Wuhan, 430074, PR China

ARTICLE INFO

Article history:
Received 18 November 2019
Received in revised form 2 August 2020
Accepted 6 August 2020
Available online 24 August 2020

Keywords:
Scheduling problem
Total weighted earliness and tardiness
Due date
Idle time insertion method
Meta-heuristic algorithm

ABSTRACT

Distributed Permutation Flowshop Scheduling Problem (DPFSP) has become a research hotspot in recent years. However, as a service level objective, the Total Weighted Earliness and Tardiness (TWET) has not been addressed so far. Due to the importance of the service level objective in modern industry, we deal with the minimization of the TWET for the DPFSP with due windows . An Iterated Greedy (IG) algorithm, namely IG with Idle Time insertion Evaluation (IG_{ITE}), is proposed. In the algorithm, an adapted NEH heuristic with five rules based on the unit earliness weight and unit tardiness weight, the due date, and the smallest slack on the last machine is used to generate an initial solution. Destruction procedure with a dynamic size is provided to enhance the exploration capability of the algorithm. Idle time insertion method is utilized to make the completion time of jobs within the due windows or as close to the due windows as possible. A large number of experiments show that the presented algorithm performs significantly better than the five competing algorithms adapted in the literature. The performance analysis shows that the IG_{ITE} is the most appropriate for the DPFSP with due windows among the tested algorithms.

1. Introduction

In today's decentralized and globalized economy, a multi-factory environment is becoming increasingly important. Therefore, production scheduling in a multi-factory environment, namely distributed scheduling problem, has attracted more and more attention in recent years [1,2]. Among all types of distributed scheduling problems, the Distributed Permutation Flowshop Scheduling Problem (DPFSP) has become a research hotspot [3,4]. Makespan, total flowtime and total weighted earliness and tardiness (TWET) are commonly used criteria in the scheduling literature. For the DPFSP, the makespan criterion and the total flowtime criterion have been addressed in the literature [5,6]. However, another important criterion, the weighted total earliness and tardiness, to our best knowledge, has not been studied for the DPFSP so far. In the actual production environment, the completion of products in advance (namely earliness) will result in inventory and financial burden; delays in completing the product (namely tardiness) will result in liquidated damages [7]. Therefore, in this paper, we deal with the minimization of the TWET for the DPFSP with Due Windows (DPFSPDW for short).

The due window of a job is a generalization of the traditional due date, which is the time interval for expecting to complete a job [8]. The TWET with due windows can be abbreviated as $TWET^{dw}$ and the DPFSPDW can be denoted as $DPFSP|\ prmu|\ TWET^{dw}$ according to the three-field notation of Graham et al. [9]. Considering that the single machine problem with due-windows is already NP-Hard [7], the DPFSPDW is also NP-Hard because it includes f identical factories and each factory consists of a set of m machines. Therefore, meta-heuristic algorithms are among the most promising algorithms for solving the DPFSPDW.

Among all types of meta-heuristic algorithms, Iterated Greedy (IG) is a simple but powerful algorithm for solving many optimization problems [10,11]. Recently, IG has been successfully applied to the DPFSP with both makespan and total flowtime criteria [5,6]. Therefore, in this paper, we propose an IG algorithm, namely IG with idle time insertion evaluation (IG_{ITE}), to solve the DPFSPDW. First, we adapt the NEH heuristic of Nawaz, Enscore and Ham [12] to our problem based on the unit earliness weight and unit tardiness weight, the due date, and the smallest slack on the last machine. In the initial solution generation phase, we present three simple rules, namely the unit Earliness and

* Corresponding author at: School of Mechatronic Engineering and Automation, Shanghai University, Shanghai 200072, PR China.
E-mail addresses: jingxuelei@lcu.edu.cn (X.-L. Jing), panquanke@shu.edu.cn (Q.-K. Pan), gaoliang@mail.hust.edu.cn (L. Gao), yulongwang@shu.edu.cn (Y.-L. Wang).

https://doi.org/10.1016/j.asoc.2020.106629
1568-4946/

Nomenclature

n	number of jobs
j	job index
N	set of jobs
m	number of machines
i	machine index
M	set of machines
f	number of factories
k	factory index
F	set of factories
$p_{j,i}$	processing time of job$j$ on machine i
d_j^-	the earliest delivery date of job j
d_j^+	the latest delivery date of job j
$\left[d_j^-, d_j^+\right]$	due window of job j
$C_{j,i}$	completion time of job j on machine i
C_j	completion time of job j on the last machine
E_j	earliness index of job j with due windows
T_j	tardiness index of job j with due windows
W_j^E	unit earliness weight of job j
W_j^T	unit tardiness weight of job j
$TWET^{dw}$	total weight of earliness and tardiness with due windows
DPFSP	Distributed Permutation Flowshop Scheduling Problem
TWET	Total Weighted Earliness and Tardiness
IG	Iterated Greedy algorithm
IG_{ITE}	IG with Idle Time insertion Evaluation
NEH	heuristic algorithm presented by Nawaz, Enscore, Ham
DPFSPDW	DPFSP with Due Windows
EDD	the Earliest Due Date rule
LPT	the Largest Processing Time rule
LSL	the smallest slack on the last machine rule
WET	the unit Earliness and Tardiness Weight rule
EDDWET	the rule based on EDD and WET
LSLWET	the rule based on LSL and WET
NEH2	NEH adapted for DPFSP by Naderi and Ruiz
NEH2_en	NEH2 adapted for DPFSP by Ruiz, Pan, Naderi
ANEH	adaptation for DPFSPDW of the NEH2_en
NBM	Net Benefit of Movement
RDI	Relative Deviation Index
DOE	Design of Experiments
ANOVA	Analysis of Variance

Tardiness Weight rule (WET), the rule based on the Earliest Due Date rule (EDD) and WET (EDDWET for short), and the rule based on the smallest slack on the last machine rule (LSL) and WET (LSLWET for short). Then we provide a destruction procedure with a dynamic size to enhance the exploration capability of the algorithm. Additionally, we utilize an idle time insertion method to make the completion time of each job within the due windows or as close to the due windows as possible. Furthermore, we design a simple but very effective local search method. In the end, we present a formula for our problem in solution acceptance phase.

The rest of this paper is as follows. In Section 2, the literature is reviewed. In Section 3, the DPFSPDW is described. Section 4 proposes the IG_{ITE} algorithm in detail. Section 5 calibrates the IG_{ITE} algorithm. In Section 6, the presented IG_{ITE} algorithm and five competing algorithms adapted in the literature are compared and evaluated. Finally, the conclusion is presented in Section 7.

2. Literature review

To the best of our knowledge, the DPFSPDW has not been previously addressed in the literature. Therefore, we review the relevant contributions, including the DPFSP and the flowshop scheduling problems with due windows.

2.1. Distributed flowshop scheduling problem

For the DPFSP, two criteria, makespan and total flowtime, have been widely addressed in the literature. For the makespan criterion, Naderi and Ruiz [3] first proposed two factory-assignment rules (denoted as NEH1 and NEH2) together with 14 heuristic methods and two variable neighborhood decent methods (denoted as VND_a and VND_b), and provided a total of 720 large instances to test the performance of the proposed methods. Liu and Gao [13] proposed an electromagnetism-like mechanism (EM) algorithm and improved 151 solutions for the 720 instances with significantly increased CPU time. Gao and Chen [14] proposed a hybrid genetic algorithm (HGA) and updated 692 best-known solutions out of the 720 instances. Gao and Chen [15] proposed an NEH-based heuristic algorithm with a novel insertion rule to assign a group of jobs to factories at a time. Gao et al. [16] proposed a modified variable neighborhood descent (VND) algorithm with a mixture of a VND method (denoted as VND(B&B)) and an improved NEH heuristic. Gao et al. [17] presented a tabu search algorithm (TS) by generating neighborhood solutions with exchange of subsequences between factories. Lin et al. [18] presented a modified IG (IG_Lin), which produced better results than HGA and TS with significantly reduced CPU time. Wang et al. [19] proposed an estimation of distribution algorithm (EDA) by using explicit probability distributions during the search process. Naderi and Ruiz [20] proposed a scatter search algorithm (SS), which performed much better than HGA, IG_Lin, EM, TS, VND_a, VND_b and VND(B&B). Xu et al. [21] proposed a hybrid immune algorithm (HIA) and updated 585 best-known solutions out of the 720 instances. Fernandez-Viagas and Framinan [22] presented a bounded-search IG (IG_FF), which performed much better than TS, IG_Lin, and EDA. Bargaoui et al. [23] proposed a chemical reaction optimization to compare with VND_a, NEH2 and IG_FF, and updated more than 200 best-known solutions. Ruiz et al. [6] presented a two-stage IG, which improved the results of the HIA, SS and IG_FF, and updated 497 best-known solutions out of the 720 instances.

For the total flowtime criterion, Fernandez-Viagas et al. [24] presented eighteen constructive heuristics to obtain high-quality solutions in reasonable time and an evolutionary algorithm (EA) to further optimize the solutions obtained by the heuristics. Pan et al. [5] proposed three constructive heuristics and four meta-heuristics including a discrete artificial bee colony (DABC), scatter search, iterated local search, and IG. According to the experiments conducted by the authors, both the proposed heuristics and meta-heuristics improved the existing heuristics and meta-heuristics in the literature, respectively.

From the above review, we can see that the DPFSP has become a hotspot. The authors paid more attention to the makespan and total flowtime criteria. However, the criterion of weighted earliness and tardiness has not been addressed, let alone the DPFSP with due windows. Although there are many similarities between our problem and those studied in the above literature, including the mathematical model, heuristic information, solution representation, assignment rules, operators, and algorithmic frameworks, due to the problem-specific characteristics, the existing methods have to be well tailored and new knowledge has to be explored to design an effective algorithm for solving the DPFSPDW under consideration.

2.2. Flowshop scheduling problem with due windows

As for the scheduling problems related to due windows, Jolai et al. [25] proposed a genetic algorithm to maximize the profit for no-wait hybrid flowshop with due window and job rejection. Sheikh [26] presented a genetic algorithm to maximize the total profit and minimize deviation from due date for a flexible flow line scheduling problem with time lag and due windows. Pan et al. [7] proposed four algorithms based on IG and iterated local search algorithms to minimize earliness and tardiness for a hybrid flowshop scheduling with due windows. Wang and Li [27] studied a bicriterion of due window assignment cost and the total resource consumption cost for a single-machine scheduling with a negotiable common due window, and provided an optimal algorithm by solving a series of mixed integer linear programming models and a two-dimensional fully polynomial-time approximation scheme for the Pareto set. Zhang et al. [28] proposed an idle time insertion algorithm for optimizing a given order sequence and developed three algorithms based on iterated local search, simulated annealing and tabu search for an order scheduling problem with time windows. Khare and Agrawal [29] presented a hybrid squirrel search algorithm, an opposition based whale optimization algorithm, and a discrete grey wolf optimization for the hybrid flowshop scheduling with sequence-dependent setup times to minimize TWET. Vaez et al. [30] presented a bi-objective model of maximizing the total profit and minimizing CO_2 emissions to solve a lot-sizing and scheduling problem with delivery time window and sequence-dependent setup cost consideration.

From the above review, we can see that various kinds of scheduling problems with due windows have been researched in the literature. The authors provided many methods to deal with the deviation of the job completion time from their due windows, such as inserting idle time, net benefit of movement, and penalty to the earliness and/or tardiness, which inspire us to effectively solve the DPFSPDW considered. However, the above scheduling problems with due windows are quite different from our scheduling problem. Their scheduling algorithms cannot be directly used to solve our problem due to problem-specific characteristics. Therefore, it is necessary to develop new algorithms for the problem under consideration.

3. Problem description

The DPFSPDW can be described as follows. There are a set of n jobs $N = \{1, 2, \ldots, n\}$ to be processed in a set of f factories $F = \{1, 2, \ldots, f\}$. Each job can be processed by any factory $k \in F$. Each factory is a flowshop that consists of a set of m machines $M = \{1, 2, \ldots, m\}$. In each factory, all jobs are processed in the same route, that is, first on machine 1, then on machine 2, and so on, until machine m. The processing time of job $j \in N$ on machine $i \in M$ is represented by $p_{j,i}$. The due window of job j can be denoted as $\left[d_j^-, d_j^+\right]$, where d_j^- and d_j^+ represent the earliest and latest delivery date, respectively. Without loss of generality, we assume that at any time, each machine can only serve one job at most and one job can only be processed on at most one machine. Interruption and preemption are forbidden.

Let us denote the completion time of job $j \in N$ on machine $i \in M$ as $C_{j,i}$. In particular, the completion time of job j on the last machine is represented by C_j. When C_j is less than d_j^-, it means that the job j is completed earlier than d_j^-. The earliness index is $E_j = max\left(d_j^- - C_j, 0\right)$. On the contrary, if C_j is greater than d_j^+, it means the job j is completed later than d_j^+. The tardiness index is $T_j = max\left(C_j - d_j^+, 0\right)$.

Weight is used to measure the unit cost of completing each job in advance or in delay. Let W_j^E and W_j^T represent the unit earliness and tardiness weight of job j, respectively. The total weight of earliness and tardiness with due windows is represented by $TWET^{dw} = \sum_{j=1}^{n}\left(W_j^E \times E_j + W_j^T \times T_j\right)$. The purpose of this paper is to minimize the value of $TWET^{dw}$.

Let us give an example of the DPFSPDW with two identical factories, four jobs and two machines where each factory has two machines. The unit earliness/tardiness weight, processing time and due windows are as follows.

$$\begin{pmatrix} W_j^E \\ W_j^T \end{pmatrix}_{1\times 4} = \begin{pmatrix} 5 & 1 & 3 & 2 \\ 3 & 5 & 4 & 2 \end{pmatrix},$$

$$\left(p_{j,i}\right)_{4\times 2} = \begin{pmatrix} 91 & 37 \\ 77 & 33 \\ 70 & 84 \\ 72 & 31 \end{pmatrix},$$

$$\begin{pmatrix} d_j^- \\ d_j^+ \end{pmatrix}_{1\times 4} = \begin{pmatrix} 149 & 126 & 169 & 133 \\ 178 & 154 & 176 & 148 \end{pmatrix}.$$

Fig. 1 shows a solution to this example. The processing of jobs in factories is shown in the Gantt chart. And the earliness and the tardiness are as follows.

$$\left(E_j\right)_{4\times 1} = \begin{pmatrix} 0 \\ 16 \\ 15 \\ 0 \end{pmatrix}, \left(T_j\right)_{4\times 1} = \begin{pmatrix} 20 \\ 0 \\ 0 \\ 32 \end{pmatrix}.$$

We can get objective function value: $TWET^{dw} = 3\times 20 + 1\times 16 + 3\times 15 + 2\times 32 = 185$.

4. Proposed Iterated Greedy algorithm

IG algorithm proposed by Ruiz and Stützle [10] is a simple but effective algorithm for scheduling problems. The basic IG starts from an initial solution, and then enters some local search methods to improve the solution, then iterates through destruction, reconstruction, local search (optional but not required) and solution acceptance phases. IG often uses one or more high-performance heuristics to initialize the solution, most of which use variants of the NEH algorithm. In the destruction phase, the incumbent solution is destroyed and the original permutation is divided into two parts, one consisting of the removed elements and the other consisting of the leftover elements. In the reconstruction phase, a greedy heuristic is often used to reintroduce the removed elements into the solution, forming a new, complete solution. In the local search phase, some local search methods are used to improve the solution generated in the reconstruction phase. In solution acceptance phase, a way is needed to decide whether to accept the new solution or not: it must be accepted when the new solution is better than the current best solution, otherwise, it may be accepted with a certain probability as in simulated annealing.

In this section, we propose an IG algorithm, namely IG with Idle Time insertion Evaluation (IG_{ITE}), to solve the DPFSPDW.

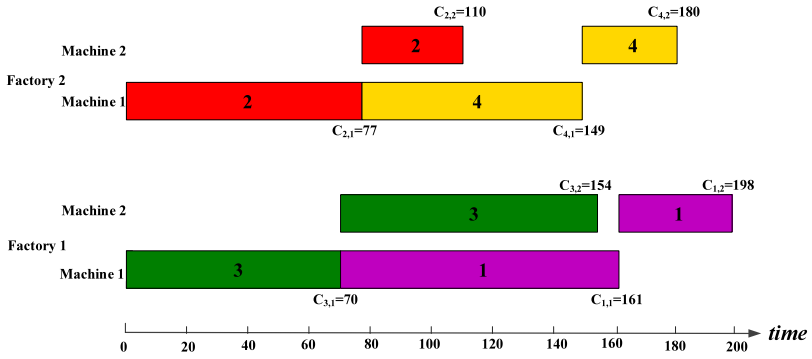

Fig. 1. A scheduling Gantt chart for the example instance.

We detail IG_{ITE} as follows: solution representation, initialization method, idle time insertion method, destruction with dynamic size, reconstruction phase, local search, solution acceptance phase and the complete procedure of IG_{ITE}.

4.1. Solution representation

We use a set of f sequences to represent a solution, one per factory. The sequence of the factory k is represented by $\pi_k = \left\{\pi_{k,1}, \pi_{k,2}, \ldots, \pi_{k,n_k}\right\}$, where $k = 1, 2, \ldots, f, j = 1, 2, \ldots, n_k$, $\pi_{k,j} \in N$ is the jth job in π_k, n_k is the total number of jobs assigned to factory k, and $\sum_{k=1}^{f} n_k = n$. The set of the f sequences, $\pi = \{\pi_1, \pi_2, \ldots, \pi_f\}$, represents a solution. For the example in Fig. 1, the solution can be represented by $\pi = \{\pi_1, \pi_2\}$ with $\pi_1 = \{3, 1\}$ and $\pi_2 = \{2, 4\}$.

4.2. Initialization method

One of the most effective heuristics to solve the permutation flowshop scheduling problem is the NEH heuristic [31]. Ruiz et al. [6] and Pan et al. [5] have already successfully extended the NEH heuristic to DPFSP with makespan and total flowtime criteria, respectively. Now, we extend the NEH heuristic to the DPFSPDW. The basic NEH heuristic first generates a seed sequence of jobs by sorting all the jobs according to the Largest Processing Time (LPT) rule, which gives high priority to the job with large processing time. However, for the DPFSPDW, the TWET objective is mainly related to the due date and the unit earliness and tardiness weight of each job. The Earliest Due Date (EDD) and the smallest slack on the last machine (LSL) rules, commonly used in the literature for scheduling problems with *TWET* criterion, would be better to yield a seed sequence of jobs for the NEH heuristic to solve the DPFSPDW. In this paper, we present three simple rules based on the unit earliness and tardiness weight of each job, EDD rule, and LSL rule.

The first rule is called WET, which first divides the jobs into two groups, G_W^T and G_W^E, according to their unit earliness weights and unit tardiness weights, where G_W^T includes all the jobs whose unit tardiness weight is larger than or equal to the unit earliness weight, and G_W^E consists of jobs whose unit tardiness weight is less than the unit earliness weight. Then the jobs in the group G_W^T are sorted according to their unit tardiness weights in non-increasing order and obtain a partial sequence τ_W^T, while the jobs in the group G_W^E are sorted in the non-decreasing order of their unit earliness weights and yield a partial job sequence τ_W^E. Finally, the two sequences are united into a whole job sequence that is used as seed sequence in the NEH heuristic. The pseudocode of WET is given in Algorithm 1.

Algorithm 1. WET(τ)

$G_W^T \leftarrow \{\}, G_W^E \leftarrow \{\}, \tau_W^T \leftarrow \{\}, \tau_W^E \leftarrow \{\}, \tau \leftarrow \{\}$
for j=1 **to** n **do** %(jobs are divided into two groups)
  if $w_j^T \geq w_j^E$ **then**
    $G_W^T \leftarrow$ job j
  else
    $G_W^E \leftarrow$ job j
  endif
endfor
$\tau_W^T \leftarrow$ Sort G_W^T according to w_j^T in non-increasing order
$\tau_W^E \leftarrow$ Sort G_W^E according to w_j^E in non-decreasing order
$\tau \leftarrow \tau_w^T + \tau_w^E$
Output τ

Algorithm 2. EDDWET(τ)

$G_W^T \leftarrow \{\}, G_W^E \leftarrow \{\}, \tau_W^T \leftarrow \{\}, \tau_W^E \leftarrow \{\}, \tau \leftarrow \{\}$
for j=1 **to** n **do** %(jobs are divided into two groups)
  if $w_j^T \geq w_j^E$ **then**
    $G_W^T \leftarrow$ job j
  else
    $G_W^E \leftarrow$ job j
  endif
endfor
$\tau_W^T \leftarrow$ Sort G_W^T according to w_j^T in non-increasing order
$\tau_W^E \leftarrow$ Sort G_W^E according to w_j^E in non-decreasing order
for i=1 **to** $min(sizeof(\tau_W^T), sizeof(\tau_W^E))$ **do**
  Compare d_j^+ of the first job from τ_W^T and the first job from τ_W^E
  Take the job with smaller value from its sequence and append to τ
endfor
if $sizeof(\tau_W^T) > 0$ **then**
  $\tau \leftarrow \tau + \tau_w^T$
endif
if $sizeof(\tau_W^E) > 0$ **then**
  $\tau \leftarrow \tau + \tau_w^E$
endif
Output τ

The second rule is called EDDWET and considers not only the unit earliness and tardiness weight, but also the due date. We first obtain two partial job sequences τ_W^T and τ_W^E like the rule WET. Then we compare the first job of each partial sequence, and take the job with the earlier due date (here we use d_j^+) out of its sequence, and append it to the sequence τ. The above process is repeated until a partial job sequence is empty. The rest job/jobs in τ_W^T or τ_W^E is/are appended to τ. The pseudocode of EDDWET is given in Algorithm 2.

The third rule, called LSLWET, considers the unit earliness weight, the unit tardiness weight and the slack on the last machine. It is almost the same as EDDWET except that we compare jobs from two partial sequences τ_W^T and τ_W^E according to $d_j^+ - p_{j,m}$ instead of d_j^+ and give higher priority to the job with smaller $d_j^+ - p_{j,m}$.

Algorithm 3. ANEH(α)

$\tau \leftarrow$ seed sequence by *rule* α
for k=1 **to** f **do** % (*Initialization* $\pi, \pi \leftarrow \{\pi_1, \pi_2, ..., \pi_f\}$)
  $\pi_k \leftarrow \{\ \}$
endfor
for *step*=1 **to** n **do**
  for k=1 **to** f **do**
    Test job τ_{step} in all possible positions of π_k
    $TWET_k^{dw}$ is the lowest $TWET^{dw}$ obtained by inserting job τ_{step} into factory k
    p^k is the position where the lowest $TWET^{dw}$ is obtained
  endfor
  $k^* \leftarrow \arg(\min_{k=1}^{f}(TWET_k^{dw}))$
  Insert job τ_{step} of π_{k^*} at position p^{k^*} resulting in the lowest $TWET^{dw}$
endfor
Output π

With the above three proposed simple rules, plus EDD and LSL, we can obtain an adapted NEH (ANEH for short) heuristic with a parameter α, where $\alpha = 1, 2, 3, 4, 5$, respectively means WET, EDDWET, EDD, LSLWET, LSL, is used to generate a seed sequence. In the presented IG_{ITE} algorithm, we will select the best parameter, α, by calibration. We give the pseudocode of ANEH in Algorithm 3. From the pseudocode, we can see that ANEH is an extension of NEH2_en to the DPFSPDW by using different rules to generate seed sequences. Because of the problem-specific characteristics, the ANEH cannot use the acceleration algorithm proposed in the literature [22].

4.3. Idle time insertion method

In order to make the $TWET^{dw}$ value of a solution as small as possible, we need to make the completion time of each job within the due windows or as close to the due windows as possible. In the literature on the scheduling problems with TWET criterion, the inserting idle time is the most commonly used method [7]. The Net Benefit of Movement (NBM) proposed by Tseng and Liao [32] is a fast method of inserting idle time on a type of flowshop lot-streaming problem. Pan et al. [7] extended the NBM method to the hybrid flowshop with due windows. Here we adapt the NBM to our problem.

We show the pseudocode for the insertion of idle time for the last stage of a factory in Algorithm 4. In fact, inserting idle time can be added at every stage of each factory. However, only the last stage is enough to evaluate a solution. Therefore, this paper only discusses inserting idle time in the last stage of each factory. Considering the machine in the last stage, we separate jobs or blocks of consecutive jobs by idle times. Let us call a block of jobs/job S_M. We can divide the jobs/job of S_M into three different subsets: $S_E = \{j | E_j > 0, j \in S_M\}$ represents jobs that are early, and $S_T = \{j | T_j \geq 0, j \in S_M\}$ represents jobs that are tardy, and $S_D = \{j | C_j \geq d_j^-, C_j < d_j^+, j \in S_M\}$ represents jobs that are on time. Considering that the minimum idle time insertion is one unit, a job j with $C_j = d_j^+$ which is on time and no tardy is included in S_T instead of S_D since it would be tardy after inserting just one unit of idle time. The earliness decreases by $\sum_{j \in S_E} W_j^E$ and the tardiness increases by $\sum_{j \in S_T} W_j^T$ when all the jobs of S_M are moved forward one unit. Therefore, if $\sum_{j \in S_E} W_j^E > \sum_{j \in S_T} W_j^T$, we can decide to insert idle time before S_M. We can actually continue to insert idle time until S_E or S_T changes. The maximum possible idle time insertion can be calculated by $\theta_1 = \min\left\{\min_{j \in S_E}\{E_j\}, \min_{j \in S_D}\{d_j^+ - C_j\}\right\}$. Insertion of more than θ_1 idle time results in that at least one job moves from S_D to S_T or S_E to S_D. In addition, there may be another job block after S_M, so we use θ_2 to indicate the idle time between the two blocks. Taking into account all of the above, the maximum idle time insertion before a block S_M is $\theta = \min\{\theta_1, \theta_2\}$. In Algorithm 4, the loop goes for all jobs included in factory k. The calculation of θ_1 also needs to traverse all the jobs included in factory k, and in the worst case, the computational complexity of the idle time insertion heuristic is $O\left(n_k^2\right)$. In Algorithm 5, the loop goes for all factories. At the extreme, if there is only one factory, the computational complexity of the idle time insertion heuristic is $O\left(n^2\right)$.

Algorithm 4. Getfit_IdleTimeInsertion(π_k)

$s \leftarrow n_k$
while $s>0$ **do**
  if there is a job to the right of S_M **then**
    Calculate idle time θ_2
  else
    $\theta_2 \leftarrow +\infty$
  endif
  Generate S_E, S_D, and S_T from S_M
  if $\sum_{j \in S_E} W_j^E > \sum_{j \in S_T} W_j^T$ **then**
    Calculate $\theta_1 \leftarrow \min\left\{\min_{j \in S_E}\{E_j\}, \min_{j \in S_D}\{d_j^+ - C_j\}\right\}$
    Insert $\theta \leftarrow \min\{\theta_1, \theta_2\}$ time units before S_M
    Set $C_{j,m} \leftarrow C_{j,m} + \theta,\ \forall j \in S_M$
    Set $S_{j,m} \leftarrow S_{j,m} + \theta,\ \forall j \in S_M$ %(Update start time)
    Set $E_j \leftarrow E_j - \theta,\ \forall j \in S_E$
    Set $T_j \leftarrow T_j + \theta,\ \forall j \in S_T$
  else
    $s \leftarrow s$-1
  endif
endwhile
$fit \leftarrow$ Calculate total penalties of factory k
Output fit

Algorithm 5. GetFit_IdleTimeInsertion(π)

for k=1 **to** f **do**
  $fit_k \leftarrow$ ***Getfit_IdleTimeInsertion***(π_k) %(use **Algorithm 4**)
endfor
$Fit \leftarrow \sum_{k=1}^{f} fit_k$
Output Fit

Now we proceed with an example with 10 jobs and 2 factories at the last stage in the DPFSPDW. Fig. 2 shows the insertion idle time process for the example. The due windows and the unit earliness/tardiness weight are as follows.

$$\begin{pmatrix} d_j^- \\ d_j^+ \end{pmatrix}_{1\times 10} = \begin{pmatrix} 307 & 274 & 249 & 278 & 266 & 259 & 265 & 279 & 304 & 284 \\ 340 & 303 & 280 & 319 & 271 & 316 & 288 & 284 & 337 & 333 \end{pmatrix},$$

$$\begin{pmatrix} W_j^E \\ W_j^T \end{pmatrix}_{1\times 10} = \begin{pmatrix} 5 & 5 & 2 & 4 & 5 & 1 & 1 & 2 & 2 & 5 \\ 4 & 4 & 5 & 3 & 1 & 1 & 1 & 3 & 3 & 2 \end{pmatrix}.$$

Fig. 2(a) shows the initial schedule, without inserting idle time method, jobs 5, 7 and 8 are early ($E_5 = 47$, $E_7 = 15$, $E_8 = 59$), jobs 2, 3 and 4 finish on time, and jobs 1, 6, 9 and 10 are tardy ($T_1 = 7$, $T_6 = 80$, $T_9 = 44$, $T_{10} = 75$). Therefore, $TWET^{dw} = 758$. Fig. 2(j) is the final schedule which is after inserting idle time, jobs 5, 7 and 8 are early ($E_5 = 22$, $E_7 = 5$, $E_8 = 47$), jobs 3 and 4 finish on time, and jobs 1, 2, 6, 9 and 10 are tardy ($T_1 = 7$, $T_2 = 4$, $T_6 = 80$, $T_9 = 44$, $T_{10} = 75$). Therefore, $TWET^{dw} = 615$.

The procedure begins with the last job in each factory. Fig. 2 (b) shows job 10 is tardy, job 4 and job 2 are on time, none of the three jobs can be considered. Job 5 is early, it can be considered, $S_M = \{5\}$, job 2 is to the right of job 5, and the idle time is $\theta_2 = 223 - 219 = 4$, $\theta_1 = 266 - 219 = 47$, $\theta = \min\{\theta_1, \theta_2\} = 4$. After inserting 4 units of idle time before job 5, the resulting schedule is shown in Fig. 2 (c), $S_M = \{5, 2\}$, $E_5 = 43$, $S_E = \{5\}$ and $S_D = \{2\}$, we can also continue to insert idle time, $\theta_1 = \min\{43, 303 - 286\} = 17$, $\theta_2 = 299 - 286 = 13$, $\min\{\theta_1, \theta_2\} = 13$; after inserting 13 units of idle time before job 5, the resulting schedule is shown in Fig. 2(d), $S_M = \{5, 2, 4\}$, $E_5 = 30$, $S_E = \{5\}$ and $S_D = \{2, 4\}$, we can also continue

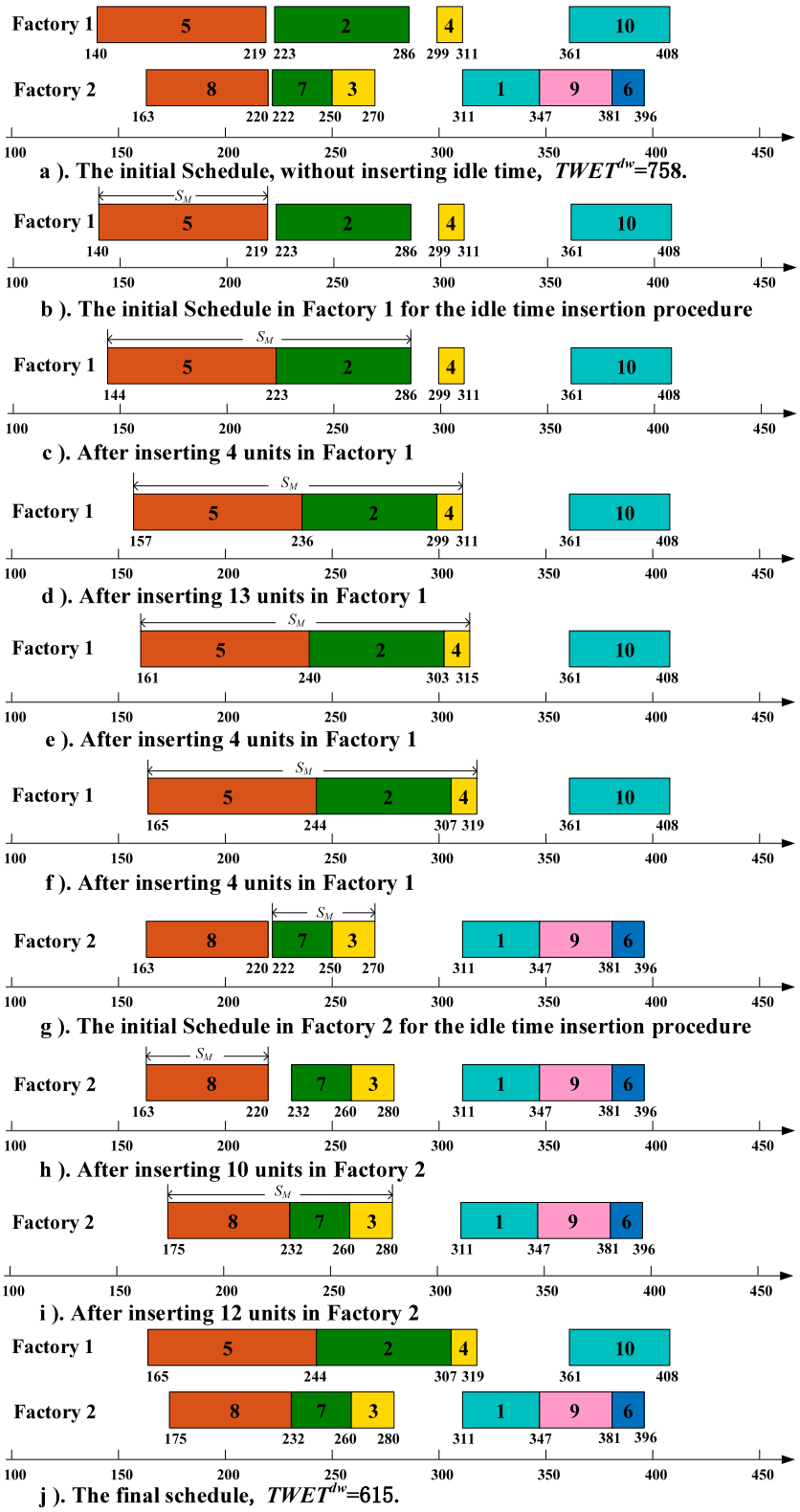

Fig. 2. Gantt charts for the example application of the idle time insertion procedure.

to insert idle time, $\theta_1 = \min\{30, 303-299, 319-311\} = 4$, $\theta_2 = 361-311 = 50$, $\theta = \min\{\theta_1, \theta_2\} = 4$; after inserting 4 units of idle time before job 5, the resulting schedule is shown in Fig. 2(e), $E_5 = 26$, $T_2 = 0$, $S_E = \{5\}$, $S_T = \{2\}$ and $S_D = \{4\}$, we can also continue to insert idle time because $W_5^E > W_2^T$, $\theta_1 = \min\{26, 319-311-4\} = 4$, $\theta_2 = 361-311 = 50$, $\theta = \min\{\theta_1, \theta_2\} = 4$, after inserting 4 units of idle time before job 5, the resulting schedule is shown in Fig. 2(f) with an improvement in the $TWET^{dw}$ of 109 units. Fig. 2(g) shows jobs 6, 9 and 1 are tardy, job 3 is on time, none of the four jobs can be considered, job 7 is early, so it can be considered, and we can see jobs 7and 3 form a new block of jobs, $S_M = \{7, 3\}$, and the idle time

is $\theta_1 = \min\{15, 280-270\} = 10$, $\theta_2 = 311-270 = 41$, $\theta = \min\{\theta_1, \theta_2\} = 10$. After inserting 10 units of idle time before job 7, the resulting schedule is shown in Fig. 2(h), $S_M = \{7, 3\}$ can no longer insert idle time because $W_7^E < W_3^T$, and job 8 is considered. Now, $S_M = \{8\}$, $E_8 = 59$, and the idle time is $\theta_1 = 59$, $\theta_2 = (222+10)-220 = 12$, $\theta = \min\{\theta_1, \theta_2\} = 12$. After inserting 12 units of idle time before job 8, the resulting schedule is shown in Fig. 2 (i), $E_8 = 47$, $E_7 = 5$, $T_3 = 0$, $S_M = \{8, 7, 3\}$, $S_E = \{8, 7\}$, $S_T = \{3\}$, $S_M = \{8, 7, 3\}$ can no longer insert idle time because $W_8^E + W_7^E < W_3^T$, the resulting schedule is shown in Fig. 2(i) with an improvement in the $TWET^{dw}$ of 34 units.

As a matter of fact, we present the IG algorithm, IG_{ITE}, which performs the idle time insertion method to each solution or partial solution generated during reconstruction phase and local search phase. Furthermore, we also perform the idle time insertion method to the initial solution generated by the NEH-based heuristics.

4.4. Destruction with dynamic size

During the destruction phase, a number of d jobs are randomly selected and removed from the incumbent solution π. The rest of the solution is expressed as π^{rest}. The extracted d jobs form a sequence γ. In the literature, parameter d is usually a predefined constant and it does not change during the iterations. However, in our primary experiment, we find that the d value changing dynamically in a given range leads to a better IG algorithm. This is mainly because different d values lead to more diverse search areas than a single d value does. With the idea of developing a simple but effective IG algorithm, we randomly generate a d value for the destruction using a uniform distribution in the range of [1, d^{max}], where d^{max} is a parameter that is calibrated experimentally. The pseudocode of the destruction is given in Algorithm 6.

Algorithm 6. Destruction(π, d^{max})

```
π^rest ← π
γ ← { }
d ← rand( )%d^max +1
while sizeof(γ) < d do
    Randomly extract a job j from π^rest
    Append the job j to γ
endwhile
Output π^rest and γ
```

4.5. Reconstruction phase

During the reconstruction phase, a new complete solution π is reconstructed by using the greedy method to reinsert all the removed jobs. To insert job γ_s, $s = 1, 2, \ldots, d$, all the possible positions of all the factories are tested. Job γ_s is finally inserted at the position of the factory that leads to the lowest increase of $TWET^{dw}$. The pseudocode is given in Algorithm 7.

4.6. Local search

To further enhance the performance of the IG algorithm, a local search process is usually applied to the solution generated by the reconstruction process. For the job permutation-based representation, local search procedures defined by an insertion and pairwise exchange operator are commonly used in the literature. The insertion operator extracts a job from the permutation and then reinserts it into another position, whereas the pairwise exchange operator exchanges a pair of jobs at the different positions. The insertion and pairwise exchange operators are quite different. They often lead to different search space. Since in our reconstruction procedure, the operator is similar to the insertion operator, we consider the pairwise exchange operators here to keep the diversity of search process. Our local search procedure is very simple. For the current solution, in each factory with no less than two jobs, two jobs are randomly selected with uniform distribution and then their positions are exchanged. If the current solution is better than the obtained solution, the obtained solution is replaced with the current solution; otherwise, the obtained solution is not changed. The procedure is repeated until all the factories are considered. The pseudocode of the local search is given in Algorithm 8.

Algorithm 8. Localsearch_IdleTimeInsertion(π)

```
for k=1 to f do
  if sizeof(π_k)>=2 then
    fit=Getfit_IdleTimeInsertion(π_k)        %(use Algorithm 4)
    π'_k ← Perform a random pairwise exchange to π_k
    Tfit=Getfit_IdleTimeInsertion(π'_k)      %(use Algorithm 4)
    if Tfit< fit then
      π_k ← π'_k
    endif
  endif
endfor
Output π
```

4.7. Solution acceptance phase

During the solution acceptance phase, it is decided whether the new solution generated by the local search becomes the starting point for the next iteration [33]. For most of IG algorithms in the literature, the simulated annealing acceptance criterion with a constant temperature T was used [34]. To minimize the total tardiness in a permutation flowshop scheduling problem, Korhan Karabulut [35] presented a formula to compute temperature T as follows:

$$T=\beta \times \frac{\sum_{j=1}^{n}\left(LB_{C\max}-d_j\right)}{n\times 10} \tag{1}$$

where β is a parameter to be adjusted, and $LB_{C\max}$ is the lower bound of the makespan which are suggested by Taillard [36].

For our problem, the due date is a time window. The tardiness is generated by $C_j - d_j^+$. Further, there are no lower bounds of the makespan for the DPFSP instances under consideration published in the literature. Furthermore, we also want to develop a simple but effective algorithm. Therefore, we adapt the formula (1) to our problem by replacing d_j with d_j^+ and $LB_{C\max}$ with $C_{\max}^{NEH}$ as follows.

$$T = \beta \times \frac{\sum_{j=1}^{n}(C_{\max}^{NEH}-d_j^+)}{n\times 10} \tag{2}$$

where $C_{\max}^{NEH}$ is the *makespan* generated by NEH2 [3] using the LPT rule.

4.8. Iterated Greedy algorithm with idle time insertion evaluation

The pseudocode of the IG_{ITE} is given in Algorithm 9, where π^* represents the best solution, Fit^* represents the best target function value.

5. Experimental algorithm calibration

We calibrate the proposed algorithm using a Design of Experiments (DOE) and Analysis of Variance (ANOVA) methods that have been widely used in recent scheduling literature [37,38]. We randomly pick up 72 instances from the 720 test instances

Algorithm 7. Reconstruction_IdleTimeInsertion(π^{rest}, γ)

$\pi \leftarrow \pi^{rest}$
for s=1 to *sizeof*(γ) **do**
  for k=1 to f **do**
    $Initfit_k \leftarrow$ ***Getfit_IdleTimeInsertion***(π_k) %(use **Algorithm 4**)
    $\pi_k^{'} \leftarrow$ Test job γ_s at all the possible positions of factory k
    $\Delta_k \leftarrow$ ***min(Getfit_IdleTimeInsertion***($\pi_k^{'}$)-*Initfit*$_k$*)* , the lowest increase of $TWET^{dw}$
    p^k is the position where the lowest Δ_k is obtained in factory k
  endfor
$k* \leftarrow \arg\left(\min_{k=1,2,\ldots,f} \Delta_k\right)$
Insert job γ_s at position p^{k^*} of π_{k*}
endfor
Output π

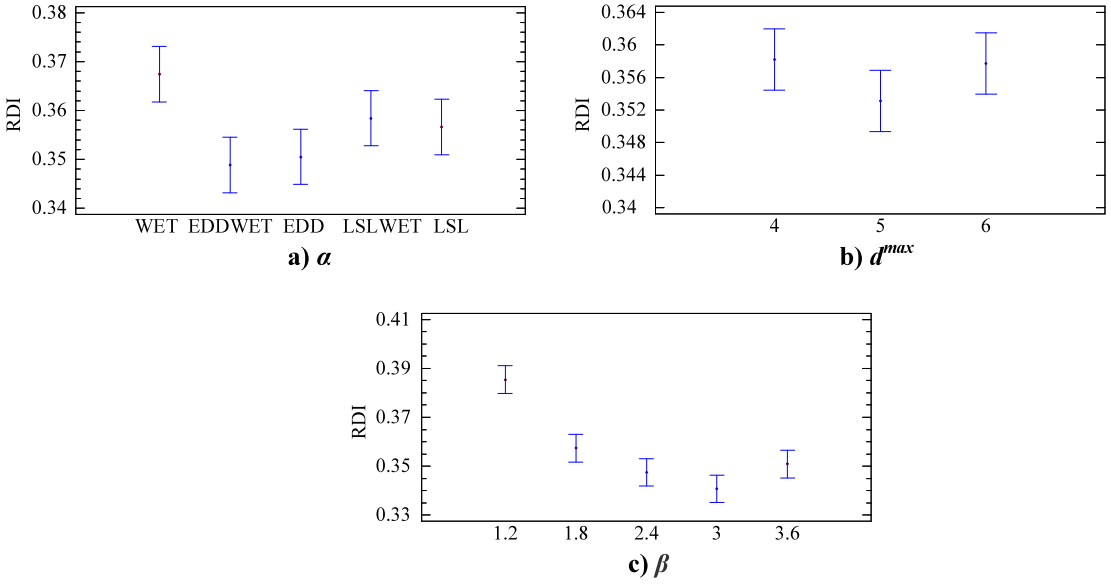

Fig. 3. Means plots for all the factors for the IG_{ITE} calibration.

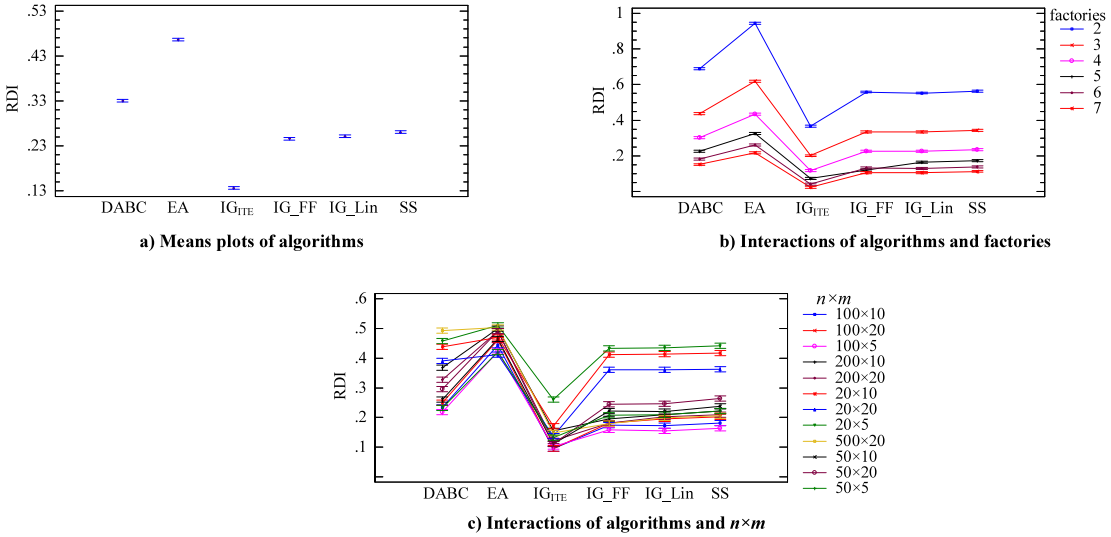

Fig. 4. Means plots and Interactions for the competing algorithms at 30*mn* milliseconds.

Table 1
ARDI for 30*mn* milliseconds (factories 2, 3, and 4; minimum ARDI values are in bold).

factories	jobs	$n\times m$	IG_{ITE}	IG_Lin	IG_FF	EA	DABC	SS
$f=2$	20	20×5	**0.664**	0.843	0.842	0.944	0.883	0.853
		20×10	**0.487**	0.829	0.827	0.929	0.883	0.834
		20×20	**0.527**	0.834	0.833	0.929	0.892	0.834
	50	50×5	**0.384**	0.552	0.552	0.944	0.585	0.571
		50×10	**0.329**	0.540	0.540	0.943	0.584	0.551
		50×20	**0.272**	0.545	0.525	0.941	0.598	0.542
	100	100×5	**0.294**	0.413	0.434	0.924	0.512	0.427
		100×10	**0.278**	0.433	0.451	0.950	0.517	0.441
		100×20	**0.239**	0.445	0.456	0.953	0.520	0.444
	200	200×10	**0.312**	0.420	0.434	0.960	0.698	0.444
		200×20	**0.260**	0.405	0.415	0.955	0.635	0.418
	500	500×20	**0.346**	0.376	0.397	0.968	0.951	0.398
		Mean	**0.366**	0.553	0.559	0.945	0.688	0.563
$f=3$	20	20×5	**0.437**	0.578	0.576	0.669	0.606	0.590
		20×10	**0.280**	0.542	0.542	0.610	0.576	0.545
		20×20	**0.235**	0.497	0.497	0.588	0.561	0.500
	50	50×5	**0.200**	0.305	0.303	0.576	0.328	0.315
		50×10	**0.162**	0.296	0.292	0.590	0.343	0.311
		50×20	**0.130**	0.298	0.290	0.590	0.352	0.311
	100	100×5	**0.155**	0.222	0.228	0.592	0.289	0.233
		100×10	**0.144**	0.246	0.245	0.603	0.321	0.256
		100×20	**0.125**	0.254	0.248	0.597	0.316	0.259
	200	200×10	**0.196**	0.264	0.276	0.670	0.457	0.276
		200×20	**0.157**	0.255	0.263	0.650	0.412	0.263
	500	500×20	**0.186**	0.247	0.252	0.687	0.675	0.259
		Mean	**0.201**	0.334	0.334	0.618	0.436	0.343
$f=4$	20	20×5	**0.239**	0.389	0.387	0.470	0.408	0.393
		20×10	**0.174**	0.407	0.406	0.480	0.434	0.411
		20×20	**0.048**	0.283	0.283	0.330	0.306	0.287
	50	50×5	**0.107**	0.176	0.173	0.377	0.196	0.185
		50×10	**0.091**	0.194	0.194	0.433	0.232	0.207
		50×20	**0.117**	0.225	0.223	0.473	0.288	0.246
	100	100×5	**0.087**	0.135	0.131	0.394	0.192	0.141
		100×10	**0.081**	0.150	0.147	0.404	0.205	0.157
		100×20	**0.079**	0.172	0.166	0.413	0.225	0.180
	200	200×10	**0.146**	0.196	0.207	0.477	0.358	0.204
		200×20	**0.110**	0.187	0.196	0.475	0.305	0.194
	500	500×20	**0.124**	0.180	0.189	0.466	0.459	0.197
		Mean	**0.117**	0.224	0.225	0.433	0.301	0.234

described in Section 6 for our calibration. We carefully determine the range of parameters not only from the existing literature but also from our past experience. The proposed algorithm contains three parameters α, d^{max} and β. For the IG_{ITE} algorithm, the parameter α at five levels: WET, EDDWET, EDD, LSLWET and LSL; the other two parameters and their levels are shown as follows: d^{max} at three levels: 4, 5 and 6; β at five levels: 1.2, 1.8, 2.4, 3 and 3.6. The above factors result a total of $5\times3\times5=75$ different configurations for the IG_{ITE} algorithm.

We code the proposed algorithms using C++ in Microsoft Visual Studio 2015. For each configuration, the 72 calibration instances are solved. For each instance, five independent replicates are performed. All algorithms run on the same Intel (R) Core(TM) i7-7700 CPU @ 3.60 GHz with 8.00 GB RAM in the 64-bit Windows 7 professional Operation System, and they stop when the termination criterion of the maximum elapsed CPU time of 10*mn* milliseconds is met.

The Relative Deviation Index (RDI) is calculated as $RDI\left(TWET_i^{dw}\right) = \left(TWET_i^{dw} - TWET_{best}^{dw}\right)/\left(TWET_{worst}^{dw} - TWET_{best}^{dw}\right)$, where $TWET_i^{dw}$ is the objective value of a solution , $TWET_{best}^{dw}$ is the best objective value among the 375 results, and $TWET_{worst}^{dw}$ is the worst value among the 375 results. In special cases, when $TWET_{worst}^{dw} = TWET_{best}^{dw}$, *i.e.* all algorithms have the same solution results, set $RDI\left(TWET_i^{dw}\right) = 0$. The RDI value is between 0 and 1; the smaller the RDI value, the closer it is to the best solution; the value of 0 is the best and the value of 1 is the worst. Each instance gets $75\times5=375$ results for the IG_{ITE}. We get a total of $375\times72=27{,}000$ RDI values for the IG_{ITE} algorithm.

All results are analyzed by ANOVA, where n and m are considered to be non-controllable factors. For the reason of space, we briefly review the results of analysis and calibration. For the IG_{ITE} algorithm, α and β are statistically significant at a 95% confidence level, while d^{max} is not. The factors are set as follows: α =EDDWET, $d^{max}=5$ and $\beta=3$. Means plots for the ANOVA large calibration experiment in Fig. 3 show the factors for the IG_{ITE}.

6. Computational evaluation

Since the earliness and tardiness minimization for the DPFSP with due windows has not been studied in the literature and no ready-made test cases are available, we adapt the 720 test instances presented by Naderi and Ruiz (2010) for the DPFSP which is available at http://soa.iti.es by adding the due date window for each job. The 720 test instances include six factory configurations, *i.e.*, $f=2$, 3, 4, 5, 6 and 7, respectively,

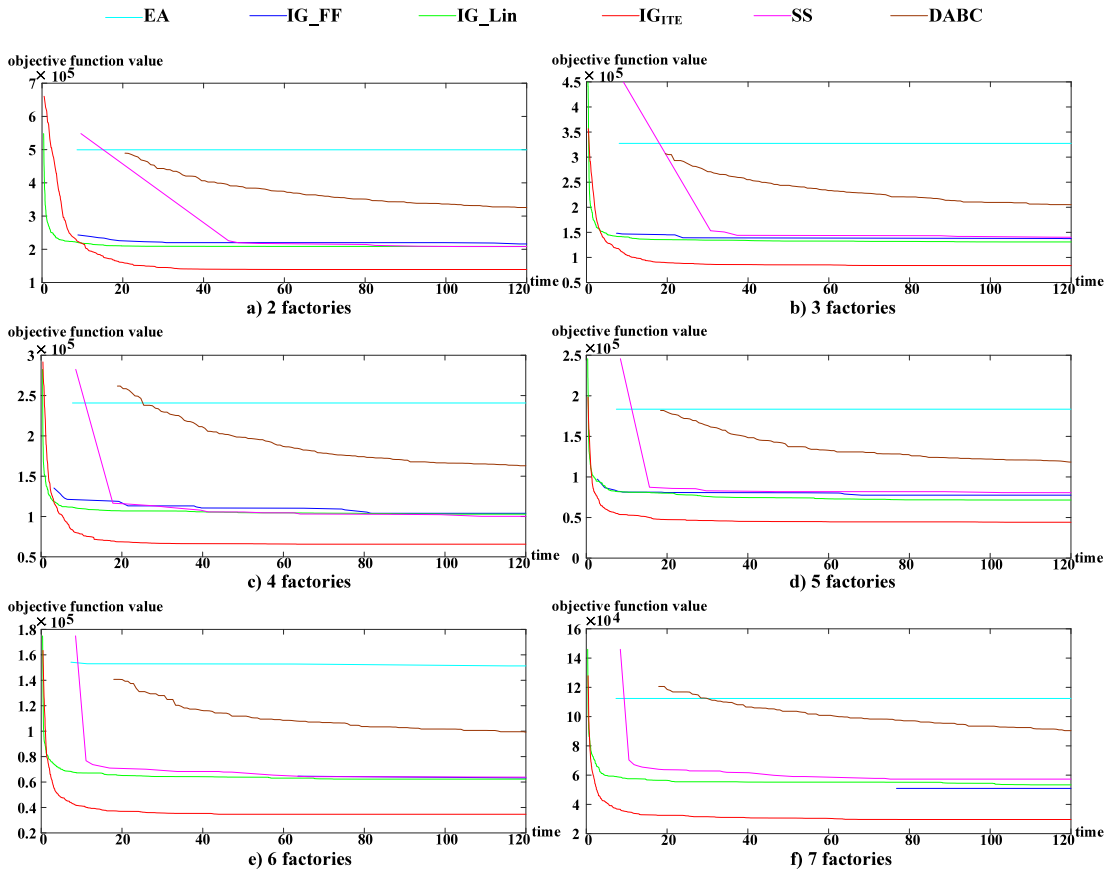

Fig. 5. Evolution of solutions for the test instance 106 with different number of factories.

Algorithm 9. IG$_{ITE}$(α, d^{max}, β)

$\pi \leftarrow$ ***ANEH***(α) % (*Initialization using* **Algorithm 3** *by rule* α)
$Fit \leftarrow$ ***GetFit_IdleTimeInsertion***(π) %(use **Algorithm 5**)
$\pi^* \leftarrow \pi$
$Fit^* \leftarrow Fit$
$T = \beta \times \frac{\sum_{j=1}^{n}(C_{\max}^{NEH} - d_j^+)}{n \times 10}$ %(use **formula (2)**)
while (time limit is not exceeded) **do**
  $\pi' \leftarrow$ ***Destruction***(π, d^{max}) %(use **Algorithm 6**)
  $\pi'' \leftarrow$ ***Reconstruction_IdleTimeInsertion***(π', γ) %(use **Algorithm 7**)
  $\pi''' \leftarrow$ ***LocalSearch_IdleTimeInsertion***(π'') %(use **Algorithm 8**)
  $TempFit \leftarrow$ ***GetFit_IdleTimeInsertion***(π''') %(use **Algorithm 5**)
  if $TempFit < Fit$ **then**
    $\pi \leftarrow \pi'''$
    $Fit \leftarrow TempFit$
    if $Fit < Fit^*$ **then**
      $\pi^* \leftarrow \pi$
      $Fit^* \leftarrow Fit$
    endif
  else if *rand() / RAND_MAX<exp{-(TempFit-Fit)/T}* **then**
      $\pi \leftarrow \pi'''$
      $Fit \leftarrow TempFit$
    endif
  endif
endwhile
Output π^* and Fit^*

each factory configuration with the 120 test instances; there are 12 combinations of n jobs and m machines, *i.e.*, $n \times m$: {20×5, 20×10, 20×20, 50×5, 50×10, 50×20, 100×5, 100×10, 100×20, 200×10, 200×20, 500×20}, and 10 instances for each combination. The unit earliness and tardiness weight of each job are generated by the random distribution of U [1,5]. The mean

Table 2
ARDI for 30mn milliseconds (factories 5, 6, and 7; minimum ARDI values are in bold).

factories	jobs	$n\times m$	IG_{ITE}	IG_Lin	IG_FF	EA	DABC	SS
$f=5$	20	20×5	**0.162**	0.328	0.328	0.395	0.343	0.334
		20×10	**0.066**	0.282	0.282	0.322	0.301	0.285
		20×20	**0.009**	0.190	0.190	0.217	0.206	0.191
	50	50×5	**0.062**	0.109	0.106	0.269	0.128	0.120
		50×10	**0.049**	0.130	0.132	0.336	0.168	0.151
		50×20	**0.084**	0.184	0.184	0.385	0.229	0.204
	100	100×5	**0.047**	0.084	0.080	0.277	0.139	0.089
		100×10	**0.044**	0.101	0.098	0.302	0.155	0.108
		100×20	0.053	0.121	**0.047**	0.321	0.167	0.131
	200	200×10	0.112	0.153	**0.000**	0.364	0.280	0.159
		200×20	0.081	0.143	**0.000**	0.353	0.238	0.149
	500	500×20	0.095	0.155	**0.000**	0.356	0.347	0.155
		Mean	**0.072**	0.165	0.121	0.325	0.225	0.173
$f=6$	20	20×5	**0.053**	0.255	0.256	0.319	0.276	0.261
		20×10	**0.011**	0.216	0.216	0.250	0.227	0.219
		20×20	**0.000**	0.189	0.190	0.218	0.202	0.193
	50	50×5	**0.033**	0.074	0.071	0.201	0.094	0.085
		50×10	**0.026**	0.094	0.098	0.270	0.133	0.115
		50×20	**0.052**	0.133	0.138	0.306	0.174	0.157
	100	100×5	**0.023**	0.051	0.053	0.206	0.105	0.058
		100×10	**0.020**	0.065	0.063	0.226	0.113	0.075
		100×20	**0.042**	0.097	0.095	0.265	0.144	0.107
	200	200×10	**0.091**	0.124	0.151	0.290	0.224	0.131
		200×20	**0.067**	0.120	0.118	0.284	0.202	0.127
	500	500×20	**0.076**	0.133	0.134	0.293	0.284	0.125
		Mean	**0.041**	0.129	0.132	0.261	0.182	0.138
$f=7$	20	20×5	**0.001**	0.213	0.213	0.264	0.230	0.216
		20×10	**0.003**	0.202	0.202	0.230	0.212	0.204
		20×20	**0.000**	0.168	0.168	0.191	0.182	0.170
	50	50×5	**0.004**	0.044	0.041	0.159	0.061	0.052
		50×10	**0.007**	0.070	0.074	0.219	0.107	0.090
		50×20	**0.012**	0.097	0.103	0.254	0.133	0.121
	100	100×5	**0.001**	0.026	0.023	0.148	0.070	0.032
		100×10	**0.002**	0.042	0.039	0.177	0.088	0.050
		100×20	**0.037**	0.078	0.078	0.221	0.121	0.089
	200	200×10	**0.075**	0.104	0.102	0.242	0.191	0.111
		200×20	**0.056**	0.100	0.099	0.233	0.174	0.108
	500	500×20	**0.063**	0.106	0.108	0.248	0.239	0.105
		Mean	**0.022**	0.104	0.104	0.215	0.151	0.112
Overall		Mean	**0.136**	0.252	0.246	0.466	0.330	0.260

due date of job j is generated according to [7] as

$$d_j = max\left(0,\ U\left[P\times(1-T-R/2),\ P\times(1-T+R/2)\right]\right), \quad (3)$$

where P is the makespan calculated by NEH2 procedure of Naderi and Ruiz [4] using LPT rule, T is tardiness factor, R is due date range. In this paper, we choose T and R equal to 0.2. Based on the d_j, we generate

$$d_j^- = max\left(d_j\times(1-H/100),\ SUM\left(p_{j,\,i}\right)\times(1+H/100)\right), \quad (4)$$

$$d_j^+ = max\left(d_j\times(1+H/100),\ SUM\left(p_{j,\,i}\right)\times(1+(3\times H)/100)\right), \quad (5)$$

where $H = U[1, 10]$, $SUM\left(p_{j,i}\right)$ represents the sum of processing times of job j on all machines.

We evaluate the proposed meta-heuristic algorithm by a large number of statistical comparisons. All algorithms are used to solve the 720 instances. We compare the IG_{ITE} algorithm proposed against the five state-of-the-art meta-heuristic algorithms from the closely related literature. These are DABC [5], EA [24], IG_Lin [18], IG_FF [22], and SS [20]. We adapt them to our problem by using our objective function and sharing most codes of our IG_{ITE} algorithms, and calibrate them as in Section 5. All the algorithms use the same maximum elapsed CPU time limit of 30mn milliseconds as the termination criteria. We perform five independent runs for each algorithm and provide a stop criterion for each of the 720 instances. Tables 1 and 2 report the average results grouped by factory configuration. In the tables, the ARDI values for each cell are averaged over the 50 RDI values obtained by running 5 times for 10 instances of a $n\times m$ combination.

From Tables 1 and 2, we can see that the proposed IG_{ITE} algorithm performs much better in terms of ARDI values than the five existing algorithms. IG_{ITE} is the best algorithm that achieves the minimum ARDI value for almost all the factory configurations except IG_FF gets 4 minimum values when the number of factories is 5. Among the five existing algorithms, EA is clearly a loser. In a word, the proposed algorithm, IG_{ITE}, is the clear winner; IG_Lin, IG_FF, and SS are close to each other, and EA and DABC are the worst two algorithms.

A multifactor ANOVA is used to determine whether the differences observed in Tables 1 and 2 are indeed significant. The type of algorithms, the number of factories and the combination of $n\times m$ are considered as factors. The results of ANOVA show that the difference of response variable (ARDI) among the three factors is statistically significant at 95% confidence level. Fig. 4(a) shows the means plots of the type of algorithms. Fig. 4(b)-(c) show the interactions of the algorithms and the number of

factories and the combination of $n \times m$, respectively. These plots use a 95% confidence level and a Tukey HSD (Honestly Significant Difference) confidence interval. The Tukey HSD method was first proposed in 1953 [39]. In the plotted means, the overlapping intervals represent statistically insignificant differences.

Fig. 4(a) shows that the overall ARDI values generated by the six algorithms are statically different. From best to worst, these values are obtained by IG_{ITE}, IG_FF, IG_Lin, SS, DABC and EA, respectively. As you can see from Fig. 4(b), as the number of factories increases, the ARDI values obtained by all algorithms decrease. Fig. 4(c) shows the same conclusion as Fig. 4(a).

Next, we randomly select the same test case for each type of factories. Fig. 5 shows the evolution of the test instance 106 with different number of factories. In Fig. 5, the horizontal axis represents the time (unit: *s*), and the vertical axis represents objective function value. We can see that the objective function values decrease as the number of factories increases. The proposed meta-heuristic algorithm, IG_{ITE}, is the best algorithm; IG_Lin, IG_FF, and SS are close to each other, and EA and DABC are the worst two algorithms.

7. Conclusions

This paper has dealt with the minimization of the total weighted earliness and tardiness for the Distributed Permutation Flowshop Scheduling Problem with Due Windows (DPFSPDW). To our best knowledge, this is the first work reported on such an important scheduling problem. We proposed an Iterated Greedy algorithm, namely IG with Idle Time insertion Evaluation (IG_{ITE}). We selected five competitive meta-heuristic algorithms namely DABC, EA, IG_Lin, IG_FF, and SS from closely related scheduling literature and carefully adapted them for our problem for comparison. The parameters and operators of all the algorithms are selected through Design of Experiments and Analysis of Variance. All the algorithms use the same maximum elapsed CPU time limit of $30mn$ milliseconds as the termination criteria. Extensive comparative evaluation based on 720 test instances shows that the presented IG_{ITE} algorithm performs significantly better than the other five competing algorithms in the literature. From the best to the worst, these results are obtained by IG_{ITE}, IG_FF, IG_Lin, SS, DABC and EA, respectively. We can see that the best three algorithms are based on the IG algorithm framework. This shows that IG algorithm is suitable to solve the DPFSPDW. As for why IG_{ITE} is better than IG_FF and IG_Lin, it is mainly due to the following reasons: (1) The EDDWET rule proposed is very suitable for the initial solution generated in the ANEH algorithm. (2) The Idle time insertion method is used in the reconstruction phase and local search phase, which is effective for the final good solution. (3) During the destruction phase, the dynamic size leads to a better IG algorithm.

In the future, we try to explore the presented algorithm to closely related scheduling problems such as the distributed assembly permutation flow-shop scheduling problem [40], the sequence dependent setup times flowshop scheduling problem [41], etc. We will also try to introduce some meta-heuristic algorithms applied to other problems [42,43] into our future research. And our future research can be related to the DPFSP with sequence-dependent setup times [44] and the vehicle scheduling problem [45], both of which are recently studied by our research group.

CRediT authorship contribution statement

Xue-Lei Jing: Conceptualization, Software, Investigation, Methodology, Optimization algorithm, Writing - original draft. **Quan-Ke Pan:** Resources, Optimization algorithm, Writing - Review & Editing, Supervision. **Liang Gao:** Resources, Optimization algorithm, Review & Editing. **Yu-Long Wang:** Optimization algorithm, Review & Editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This research is partially supported by the National Natural Science Foundation of China 61973203 and 51575212, and the National Natural Science Fund for Distinguished Young Scholars of China 51825502, and Shanghai Key Laboratory of Power station Automation Technology.

References

[1] L. De Giovanni, F. Pezzella, An improved genetic algorithm for the distributed and flexible job-shop scheduling problem, European J. Oper. Res. 200 (2) (2010) 395–408.
[2] K.-C. Ying, S.-W. Lin, C.-Y. Cheng, C.-D. He, Iterated reference greedy algorithm for solving distributed no-idle permutation flowshop scheduling problems, Comput. Ind. Eng. 110 (2017) 413–423.
[3] B. Naderi, R. Ruiz, The distributed permutation flowshop scheduling problem, Comput. Oper. Res. 37 (4) (2010) 754–768.
[4] J. Deng, L. Wang, A competitive memetic algorithm for multi-objective distributed permutation flow shop scheduling problem, Swarm Evol. Comput. 32 (2017) 121–131.
[5] Q.-K. Pan, L. Gao, L. Wang, J. Liang, X.-Y. Li, Effective heuristics and metaheuristics to minimize total flowtime for the distributed permutation flowshop problem, Expert Syst. Appl. 124 (2019) 309–324.
[6] R. Ruiz, Q.-K. Pan, B. Naderi, Iterated greedy methods for the distributed permutation flowshop scheduling problem, Omega 83 (2019) 213–222.
[7] Q.-K. Pan, R. Ruiz, P. Alfaro-Fernández, Iterated search methods for earliness and tardiness minimization in hybrid flowshops with due windows, Comput. Oper. Res. 80 (2017) 50–60.
[8] A. Janiak, W.A. Janiak, T. Krysiak, T. Kwiatkowski, A survey on scheduling problems with due windows, European J. Oper. Res. 242 (2) (2015) 347–357.
[9] R.L. Graham, E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, Optimization and approximation in deterministic sequencing and scheduling: a survey, in: Annals of Discrete Mathematics, Vol. 5, Elsevier, 1979, pp. 287–326.
[10] R. Ruiz, T. Stützle, A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem, European J. Oper. Res. 177 (3) (2007) 2033–2049.
[11] J. Dubois-Lacoste, F. Pagnozzi, T. Stützle, An iterated greedy algorithm with optimization of partial solutions for the makespan permutation flowshop problem, Comput. Oper. Res. 81 (2017) 160–166.
[12] M. Nawaz Jr, E.E. Enscore, I. Ham, A heuristic algorithm for the m machine, n job flowshop sequencing problem, Omega 8111 (1) (1983) 91–95.
[13] H. Liu, L. Gao, A discrete electromagnetism-like mechanism algorithm for solving distributed permutation flowshop scheduling problem, in: 2010 International Conference on Manufacturing Automation, Hong Kong, 2010, pp. 156-163.
[14] J. Gao, R. Chen, A hybrid genetic algorithm for the distributed permutation flowshop scheduling problem, Int. J. Comput. Intell. Syst. 4 (2011) 497–508.
[15] J. Gao, R. Chen, An NEH-Based Heuristic Algorithm for Distributed Permutation Flowshop Scheduling Problems, Technical Report SRE-10-1014, College of Information Science and Technology, Dalian Maritime University, Dalian, Liaoling Province, 116026, China, 2011.
[16] J. Gao, R. Chen, W. Deng, Y.Q. Liu, Solving multi-factory flowshop problems with a novel variable neighbourhood descent algorithm, J. Comput. Inform. Syst. 8 (5) (2012) 2025–2032.
[17] J. Gao, R. Chen, W. Deng, An efficient tabu search algorithm for the distributed permutation flowshop scheduling problem, Int. J. Prod. Res. 51 (2013) 641–651.
[18] S.-W. Lin, K.-C. Ying, C.-Y. Huang, Minimising makespan in distributed permutation flowshops using a modified iterated greedy algorithm, Int. J. Prod. Res. 51 (2013) 5029–5038.
[19] S.-Y. Wang, L. Wang, M. Liu, Y. Xu, An effective estimation of distribution algorithm for solving the distributed permutation flow-shop scheduling problem, Int. J. Prod. Econ. 145 (1) (2013) 387–396.

[20] B. Naderi, R. Ruiz, A scatter search algorithm for the distributed permutation flowshop scheduling problem, European J. Oper. Res. 239 (2) (2014) 323–334.
[21] Y. Xu, L. Wang, S.Y. Wang, M. Liu, An effective hybrid immune algorithm for solving the distributed permutation flow-shop scheduling problem, Eng. Optimiz. 46 (2014) 1269–1283.
[22] V. Fernandez-Viagas, J.M. Framinan, A bounded-search iterated greedy algorithm for the distributed permutation flowshop scheduling problem, Int. J. Prod. Res. 53 (2015) 1111–1123.
[23] H. Bargaoui, O.B. Driss, K. Ghédira, A novel chemical reaction optimization for the distributed permutation flowshop scheduling problem with makespan criterion, Comput. Ind. Eng. 111 (2017) 239–250.
[24] V. Fernandez-Viagas, P. Perez-Gonzalez, J.M. Framinan, The distributed permutation flow shop to minimise the total flowtime, Comput. Ind. Eng. 118 (2018) 464–477.
[25] F. Jolai, S. Sheikh, M. Rabbani, B. Karimi, A genetic algorithm for solving no-wait flexible flow lines with due window and job rejection, Int. J. Adv. Manuf. Technol. 42 (5–6) (2009) 523–532.
[26] S. Sheikh, Multi-objective flexible flow lines with due window, time lag, and job rejection, Int. J. Adv. Manuf. Technol. 64 (9–12) (2013) 1423–1433.
[27] D.J. Wang, Z.W. Li, Bicriterion scheduling with a negotiable common due window and resource-dependent processing times, Inform. Sci. 478 (2019) 258–274.
[28] Y. Zhang, Y.R. Dan, B. Dan, H. l. Gao, The order scheduling problem of product-service system with time windows, Comput. Ind. Eng. 133 (2019) 253–266.
[29] A. Khare, S. Agrawal, Scheduling hybrid flowshop with sequence-dependent setup times and due windows to minimize total weighted earliness and tardiness, Comput. Ind. Eng. 135 (2019) 780–792.
[30] P. Vaez, F. Sabouhi, M.S. Jabalameli, Sustainability in a lot-sizing and scheduling problem with delivery time window and sequence-dependent setup cost consideration, Sustainable Cities Soc. 51 (2019) 101718.
[31] S.F. Rad, R. Ruiz, N. Boroojerdian, New high performing heuristics for minimizing makespan in permutation flowshops, Omega 37 (2) (2009) 331–345.
[32] C.-T. Tseng, C.-J. Liao, A discrete particle swarm optimization for lot-streaming flowshop scheduling problem, Eur. J. Oper. Res. 191 (2) (2008) 360–373.
[33] Q.-K. Pan, L. Gao, X.-Y. Li, F.M. Jose, Effective constructive heuristics and meta-heuristics for the distributed assembly permutation flowshop scheduling problem, Appl. Soft Comput. 81 (2019) 105492.
[34] T. Stützle, Applying Iterated Local Search To the Permutation Flow Shop Problem, Technical Report AIDA-98-04, FG Itellektik, FB Informatik, TU Darmstadt.
[35] K. Karabulut, A hybrid iterated greedy algorithm for total tardiness minimization in permutation flowshops, Comput. Ind. Eng. 98 (2016) 300–307.
[36] E. Taillard, Benchmarks for basic scheduling problems, European J. Oper. Res. 64 (2) (1993) 278–285.
[37] B. Zhang, Q.-K. Pan, L. Gao, L.L. Meng, X.Y. Li, K.K. Peng, A three-stage multi-objective approach based on decomposition for an energy-efficient hybrid flowshop scheduling problem, IEEE Trans. Syst. Man, Cybern. (2019) http://dx.doi.org/10.1109/TSMC.2019.2916088.
[38] J.-Q. Li, X.-R. Tao, B.-X. Jia, Y.-Y. Han, C. Liu, P. Duan, Z.-X. Zheng, H.-Y. Sang, Efficient multi-objective algorithm for the lot-streaming hybrid flowshop with variable sub-lots, Swarm. Evol. Comput. (2019) http://dx.doi.org/10.1016/j.swevo.2019.100600.
[39] J.W. Tukey, Some selected quick and easy methods of statistical analysis, Trans. New York Acad. Sci. 16 (2) (1953) 88–97.
[40] S.-Y. Wang, L. Wang, An estimation of distribution algorithm based memetic algorithm for the distributed assembly permutation flow-shop scheduling problem, IEEE Trans. Syst Man Cybern. Syst. 46 (2016) 139–149.
[41] Y.H. Wang, X.T. Li, A hybrid chaotic biogeography based optimization for the sequence dependent setup times flowshop scheduling problem with weighted tardiness objective, IEEE Access 5 (2017) 26046–26062.
[42] G.-G. Wang, Y. Tan, Improving metaheuristic algorithms with information feedback models, IEEE Trans. Cybern. 49 (2019) 542–555.
[43] J.-Q. Li, Y.-Q. Han, P.-Y. Duan, Y.-Y. Han, B. Niu, C.-D. Li, Z.-X. Zheng, Y.-P. Liu, Meta-heuristic algorithm for solving vehicle routing problems with time windows and synchronized visit constraints in prefabricated systems, J. Clean Prod. (2019) http://dx.doi.org/10.1016/j.jclepro.2019.119464.
[44] J.P. Huang, Q.K. Pan, L. Gao, An effective iterated greedy method for the distributed permutation flowshop scheduling problem with sequence-dependent setup times, Swarm Evol. Comput. 59 (2020) http://dx.doi.org/10.1016/j.swevo.2020.100742.
[45] W.Q. Zou, Q.K. Pan, T. Meng, L. Gao, Y.L. Wang, An effective discrete artificial bee colony algorithm for multi-AGVs dispatching problem in a matrix manufacturing workshop, Expert Systems with Applications, http://dx.doi.org/10.1016/j.eswa.2020.113675.