Applied Soft Computing 109 (2021) 107526

Contents lists available at ScienceDirect

Applied Soft Computing

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

A green scheduling algorithm for the distributed flowshop problem

Yuan-Zhen Li [a,b], Quan-Ke Pan [b,*], Kai-Zhou Gao [a,e], M. Fatih Tasgetiren [c], Biao Zhang [a], Jun-Qing Li [a,d]

[a] School of Computer Science, Liaocheng University, Shandong Liaocheng, 252059, China
[b] School of Mechatronic Engineering and Automation, Shanghai University, Shanghai 200072, China
[c] International Logistics Management Department, Yasar University, Izmir, Turkey
[d] School of Computer Science, Shandong Normal University, Jinan, 250014, China
[e] Institute of Systems Engineering, School of Business, Macau University of Science and Technology, Macau, China

ARTICLE INFO

Article history:
Received 24 December 2020
Received in revised form 18 April 2021
Accepted 16 May 2021
Available online 29 May 2021

Keywords:
Distributed permutation flowshop scheduling
NSGA-II
Multi-objective optimization
Energy efficient
Total flowtime
Total energy consumption

ABSTRACT

In recent years, sustainable development and green manufacturing have attracted widespread attention to environmental problems becoming increasingly serious. Meanwhile, affected by the intensification of market competition and economic globalization, distributed manufacturing systems have become increasingly common. This paper addresses the energy-efficient scheduling of the distributed permutation flowshop (EEDPFSP) with the criteria of minimizing both total flow time and total energy consumption. Considering the distributed and multi-objective optimization complexity, an improved NSGAII algorithm (INSGAII) is proposed. First, we analyze the problem-specific characteristics and designed new operators based on the knowledge of the problem. Second, four constructive heuristic algorithms are proposed to produce high-quality initial solutions. Third, inspired by the artificial bee colony algorithm, we propose a new colony generation method using the operators designed. Fourth, a local intensification is designed for exploiting better non-dominated solutions. The influence of parameter settings is investigated by experiments to determine the optimal parameter configuration of the INSGAII. Finally, a large number of computational tests and comparisons have been carried out to verify the effectiveness of the proposed INSGAII in solving EEDPFSP.

1. Introduction

In recent years, green manufacturing to reduce environmental pollution and energy waste has attracted more and more attention from the world [1]. Therefore, it is extremely important to consider and measure energy conservation as an important content while studying traditional economic criteria such as makespan, total flow time, etc. Green scheduling has become a hot research topic [2]. Generally, the machine can work in multiple different states, and it is a better choice in the energy-saving state. The early research of green scheduling mainly studied when and which state the machine should be switched to [3]. However, the research on green scheduling is still very limited. Efficient methods, especially meta-heuristic methods for large-scale problems, are worth studying.

In this paper, an algorithm based on NSGAII, which is a very useful algorithm in the field of multi-objective evolution, is proposed to solve the energy-efficient distributed permutation flowshop scheduling problem (EEDPFSP) with the criteria to minimize the total flow time and total energy consumption. We summarize the main contributions as follows. (1) Four initialization algorithms based on problem-specific characteristics are proposed to produce good initial solutions. (2) Seven operators for solutions, including SpeedUp, RandSpeedUp, SpeedDown, RandSpeedDown, RightShift, Insert and Swap, are proposed and efficient search algorithms are designed based on these operators. (3) The new population generation method fused with problem characteristics and solution representation is designed. (4) A local intensification algorithm is designed to enhance the local search ability of the algorithm. Finally, a large number of numerical tests prove the effectiveness of the above design and the superiority of the algorithm proposed in this paper over KCA (Knowledge-Based Cooperative Algorithm) [4], CMA (Competitive Memetic Algorithm) [5], MOEA/D [6] and NSGAII [7] algorithms.

The remainder of this paper is organized as follows. Section 2 presents a literature review of related works. In Section 3, the EEDPFSP with total flowtime criterion is formulated. Section 4 presents the proposed and improved NSGAII for solving EEDPFSP in detail. We report the computational results and comparisons in Section 5 following the parameter setting. Finally, Section 6 provides the concluding remarks and suggests some future work.

* Corresponding author.
E-mail address: Panquanke@shu.edu.cn (Q.-K. Pan).

https://doi.org/10.1016/j.asoc.2021.107526
1568-4946/

2. Literature review

The widespread application of distributed manufacturing has attracted research on the distributed permutation flowshop scheduling problem (DPFSP) [8], which is a generalization of traditional permutation flowshop scheduling problem (PFSP). The DPFSP is more difficult and tricky than PFSP because it not only deals with the job sequence, but also determines the factory assignment [9]. Researchers have proposed various algorithms for solving DPFSP with different optimization objectives. The paper by Naderi and Ruiz [8] is the first paper on DPFSP research, in which the optimization objective is to minimize makespan. An effective hybrid immune algorithm (HIA) [10] was presented to solve the DPFSP. The effectiveness of the HIA was demonstrated by comparison with some existing heuristic algorithms and the variable neighborhood descent methods. A new tabu algorithm [11] was proposed for solving this DPFSP problem. From the intensive experiments, the proposed tabu algorithm outperforms all the existing algorithms including heuristic algorithms (i.e. NEH1, NEH2, VND (a) and VND (b)) and a hybrid genetic algorithm. Shih-Wei Lin et al. [12] presented a modified iterated greedy (MIG) algorithm for this problem to minimize the maximum completion time among all the factories. The computational results show that the good performance of the proposed MIG algorithm. Many scholars have proposed various solutions to DPFSP using various meta-heuristic algorithms, including genetic algorithm (GA) [13], estimation of distribution algorithm [14], competitive memetic algorithm (CMA) [5], scatter search algorithm [15], chemical reaction optimization [16], iterated greedy [17–20] , discrete artificial bee colony [21,22], and so on. In addition, some scholars have conducted extensive research on different variants of DPFSP. The other variants of DPFSP with extensive and realistic production conditions includes distributed no-wait flowshops [23,24], distributed no-idle flowshops [25,26], distributed hybrid flowshop [27,28], distributed blocking flowshops [29,30], distributed flowshop group scheduling problems [31] and so on.

However, the papers discussed in the previous paragraph are all to minimize a single objective. Multi-objective optimization problems are very common in real life and are in a very important position. In practice, many problems are multi-objective optimization problems. Under normal circumstances, the sub-objectives of the multi-objective optimization problem are contradictory. The improvement of one sub-objective may cause performance degradation of another or several sub-objectives. In other words, it is impossible to achieve the optimal value of multiple sub-objectives simultaneously. However, only coordination and compromise can be carried out among them, so that each sub-objective is as optimized as possible. Similarly, many real-world flowshop scheduling problems are also multi-objective optimization problems, which should consider multiple conflicting objectives, such as makespan, earliness time, idle time, total flow time, and carbon emissions [32]. Based on MOEA/D, PH-MOEAD was proposed to solve the hybrid flowshop scheduling lot-streaming problems, in which four objectives are minimized, namely the penalty caused by the average sojourn time, the energy consumption in the last stage, as well as the earliness and the tardiness values [33]. Based on analyzing some structural properties, an extended NEH insertion procedure with energy-saving capability was designed to optimize the makespan and carbon emissions at the same time [34]. Li and Ma proposed a novel multi-objective discrete artificial bee colony algorithm based on decomposition, to solve PFSP with the sequence-dependent setup times to minimize makespan and total flowtime [35]. An evolutionary multi-objective robust scheduling algorithm was proposed with consideration of machine breakdown [36]. A multi-objective discrete invasive weed optimization (MODIWO) algorithm was proposed to solve a multi-objective blocking flowshop scheduling problem (MOBFSP) that minimizes the makespan and total tardiness simultaneously [37].

Some researchers have conducted on energy-efficient flowshop scheduling. Both mixed-integer linear programming and constraint programming model formulations were proposed for the energy-efficient bi-objective no-wait permutation flowshop scheduling problems (NWPFSPs) with the total tardiness and the total energy consumption minimization simultaneously [38]. The PFSP with two conflicting objectives, namely, total flowtime and total energy consumption (TEC) are studied [39]. An improved multi-objective evolutionary algorithm based on decomposition was proposed, which solves the energy-efficient PFSP with sequence-dependent setup time, in order to minimize the makespan as an economic objective and the energy consumption as a green objective [40]. An energy-aware multi-objective optimization algorithm (EA-MOA) is presented to minimize the makespan and the energy consumptions for solving the hybrid flow shop (HFS) scheduling problem with the setup energy consumptions [41]. In a recent paper, Zhang et al. studied an energy-efficient hybrid flowshop scheduling problem, which considered machines with different energy usage rates, sequence-dependent setups, and machine-to-machine transportation operations [42].

However, there are still relatively few studies on multi-objective scheduling in DPFSP, and even less on the energy-efficient DPFSP (EEDPFSP). Deng and Wang propose a competitive memetic algorithm (CMA) with two populations to solve the DPFSP with the makespan and total tardiness criteria [43]. As far as we know, the EEDPFSP is only addressed in [4,5]. Based on some properties of the problem, a competitive memetic algorithm (CMA) was proposed to solve the DPFSP to minimize the makespan and carbon emissions [5]. A knowledge-based cooperative algorithm (KCA) was presented to solve the EEDPFSP with the criteria of minimizing both makespan and total energy consumption [4]. Based on the previously acquired knowledge, several excellent search mechanisms have been designed, including a cooperative initialization scheme, a cooperative search strategy and a knowledge-based local intensification, etc. More complicated than the DPFSP [5], the EEDPFSP contains three sub-problems, namely the factory allocation of each job, the job sequence in each factory and the speed selection of each operation. Speed selection expands the search space, thereby increasing the difficulty of the solution.

Through the discussion in the previous paragraphs, it can be seen that the DPFSP is a current research hotspot. A large number of heuristic algorithms and meta-heuristic algorithms have been proposed to solve the DPFSP and its variants. In addition, due to the ubiquity of multi-objective optimization problems in real life, some papers on multi-objective optimization of the DPFSP have appeared. However, there are very few papers on the energy-efficient DPFSP. In summary, the research on multi-objective optimization of energy-efficient distributed permutation flowshop scheduling problem is very important and urgent.

3. Problem description

3.1. Multi-objective optimization problem

In this subsection, the basic knowledge of multi-objective optimization problems (MOOP) is introduced. Without loss of generality, there are m objective functions and n-dimensional decision variables. A Multi-objective optimization problem can be described as:

$$\min f(x) = \{f_1(x), f_2(x), \ldots, f_m(x)\}, \quad x \in \Omega \tag{1}$$

where $x = (x_1, x_2, \ldots, x_n)$ is a decision variable in the decision space Ω. $f(x)$ is the objective function which is composed of conflicting sub-objective functions such as f_1, f_2, f_3, etc.

Let $\boldsymbol{a}$ and $\boldsymbol{b}$ be two feasible solutions. The solution $\boldsymbol{a}$ is said to dominate solution $\boldsymbol{b}$ (denoted as $a \succ b$) if and only if $\forall l \in \{1, 2, \ldots, m\}, f_l(a) \leq f_l(b)$ and $\exists l' \in \{1, 2, \ldots, m\}, f_{l'}(a) < f_{l'}(b)$. A feasible solution $\boldsymbol{a}$ is called a non-dominant solution, if there is no other feasible solution $a' \in \Omega$ which dominates $\boldsymbol{a}$. All the non-dominated solutions form the Pareto optimal solutions set. The objective vectors corresponding to the solutions in the Pareto optimal set constitute the Pareto optimal front.

In many practical applications, the decision-makers can choose the final solution in the Pareto optimal solution set according to the nature of the specific problem. Therefore, the first step and the key to solving the multi-objective optimization problem is to find as many Pareto optimal solutions as possible. In addition, the purpose of multi-objective optimization is to obtain a non-dominated solution set with good diversity and proximity to the true Pareto optimal front.

3.2. Formulation of EEDPFSP with total flowtime criterion

The EEDPFSP problem can be described as follows. There is a set of n jobs $N = \{1, 2, \ldots, n\}$ to be processed in a set of f factories $F = \{1, 2, \ldots, f\}$. All the factories are identical and contain a flowshop with a set of m machines $M = \{1, 2, \ldots, m\}$. Each job should be processed in any one of the F factories. In addition, if a job is assigned to a certain factory, all its operations should be processed in this factory and cannot be transferred to the other factories. In each factory, all the jobs must be processed in the same route, that is, first on machine 1, then on machine 2, and so on, until machine m. For each machine, there are s ($v = 1,\ldots, s$) different processing speeds. The speed of a machine cannot be changed when the processing of a job. The operation of job $i \in N$ on the machine $j \in M$ is denoted as O_{ij}. Each operation O_{ij} has a standard processing time $t_{i,j} > 0$. The actual processing time of $O_{i,j}$ is $p_{i,j} = t_{i,j}/V_v$, when the operation $O_{i,j}$ is processed at speed V_v. Once O_{ij} is started, it must be performed without interruption. Regarding the processing speed, processing time, and energy consumption, there are the following assumptions. The higher the processing speed is, the shorter the processing time is and the more power the machine consumes. The machine cannot be shut down until all the tasks assigned to the factory have been completed. If no job is processed on machine j (standby mode), the energy consumption per unit time is SP_j. All the jobs and machines are ready at time 0. At any time, no job can be processed on multiple machines, and no machine can process more than one job. Finally, the problem is to determine the assignment of jobs to factories, the sequences of the jobs in each factory, and the processing speed of each job on the machines, so as to minimize the total flow time and energy consumption of all the jobs.

The Indices, Parameters and Decision Variables used in this paper are listed in Table 1.

The mathematical model of the EEDNFSP is formulated as follows.

$$Minimise \quad \{TFT, TEC\} \tag{2}$$

Subject to.

$$\sum_{l=1}^{n}\sum_{f=1}^{F} x_{i,l,f} = 1, \forall i \tag{3}$$

$$\sum_{i=1}^{n}\sum_{f=1}^{F} x_{i,l,f} \leq 1, \forall l \tag{4}$$

$$\sum_{v=1}^{s} z_{i,j,v} = 1, \forall i, j \tag{5}$$

$$p_{i,j} = t_{i,j} \cdot \sum_{v=1}^{s} \frac{z_{i,j,v}}{V_v}, \forall i, j \tag{6}$$

$$C_{l,j,f} = \sum_{i=1}^{n} x_{i,l,f} \cdot p_{i,j}, l = 1, j = 1, \forall f \tag{7}$$

$$C_{l,j,f} \geq C_{l,j-1,f} + \sum_{i=1}^{n} x_{i,l,f} \cdot p_{i,j}, \forall l, j, f \tag{8}$$

$$C_{l,j,f} \geq C_{l-1,j,f} + \sum_{i=1}^{n} x_{i,l,f} \cdot p_{i,j}, \forall l > 1, j, f \tag{9}$$

$$PEC_{f,j} = \sum_{i=1}^{n}\sum_{l=1}^{n}(x_{i,l,f} \cdot p_{i,j} \cdot \sum_{v=1}^{s} z_{i,j,v} \cdot PP_{f,j,v}), \forall f, j \tag{10}$$

$$SEC_{f,j} = (C_{n_f,j,f} - C_{1,j,f} - \sum_{i=2}^{n} p_{i,j} \cdot \sum_{l=1}^{n} x_{i,l,f}) \cdot SP_{f,j}, \forall f, j \tag{11}$$

$$TEC = \sum_{f=1}^{F}\sum_{j=1}^{m}(PEC_{f,j} + SEC_{f,j}) \tag{12}$$

$$TFT(f) = \sum_{i=1}^{n_f} C_{m,i,f} \tag{13}$$

$$TFT = \sum_{f=1}^{F} TFT(f) \tag{14}$$

$$x_{i,l,f} \in \{0, 1\}, \forall i, l, f \tag{15}$$

$$z_{i,j,v} \in \{0, 1\}, \forall i, j, v \tag{16}$$

$$C_{l,j,f} \geq 0, \forall l, j, f \tag{17}$$

Constraint set (3) implies that each job must be assigned to one and only one factory, and occupy only one position in the assigned flowshop. Constraint (4) ensures that at most one job is placed in each position. Constraint set (5) is the restriction that each operation $O_{i,j}$ can be processed at one speed. The actual processing time of operation $O_{i,j}$ can be obtained by (6). Please note that $C_{l,0,f}$ represents the completion time of the lth job on the 0th machine on the fth factory, so $C_{l,0,f} = 0$. All the jobs and machines are ready at time 0. The completion time of the first job in factory f on the first machine can be calculated by (7). Constraint (8) describes the relations of the completion time of a job on adjacent machines. The processing of a job on each machine can only start when the processing of the same job on the previous machine is finished. Constraint (9) describes the relations of the completion time of adjacent jobs on the same machine. Each job can start only after the previous job assigned to the same machine at the same factory is completed. When machine j in factory f is operating in processing mode and standby mode, the total energy consumption is defined as (10) and (11), respectively. The total energy consumption of a schedule is defined as (12). The total flow time of factory f is defined as (13). The total flow time objective of the schedule is defined as (14). Constraint sets (15)–(17) give the value ranges of all the variables, where binary variable $x_{i,l,f}$, represents the factory assignment for each job and the job sequence in each factory and $z_{i,j,v}$ gives the speed choice of each operation.

Table 1
Indices, parameters and decision variables.

n	The number of jobs to be processed.
m	The number of machines in each factory.
F	The number of factories.
s	The number of processing speeds.
i	Index for jobs, $i \in \{1, \ldots, n\}$.
l	Job positions in a sequence, $l \in \{1, \ldots, n\}$.
j	Index for machines, $j \in \{1, \ldots, m\}$.
f	Index for factories, $f \in \{1, \ldots, F\}$.
v	Index for speeds, $v \in \{1, \ldots, s\}$.
V_v	The vth processing speed and $V_1 < V_2 < \ldots < V_s$.
$O_{i,j}$	Operation of job i on machine j.
$t_{i,j}$	The standard processing time of $O_{i,j}$.
$p_{i,j}$	The actual processing time of $O_{i,j}$.
$PP_{f,j,v}$	The energy consumption per unit time of machine j in factory f running at speed v.
$SP_{f,j}$	The energy consumption of the machine j at stand-by mode per unit time in factory f.
$PEC_{f,j}$	Continuous variable for TEC when machine j runs at processing mode in factory f.
$SEC_{f,j}$	Continuous variable for TEC when machine j runs at stand-by mode in factory f.
TEC	Continuous variable TEC
$C_{l,j,f}$	Continuous variable for the completion time of position l on machine j in factory f.
$TFT(f)$	Continuous variable for the total flowtime of factory f.
TFT	The total flowtime of the schedule.
$x_{i,l,f}$	Binary variable that takes value 1 if job i occupies position l in factory f, and 0 otherwise.
$z_{i,j,v}$	Binary variable that takes value 1 if job i is processed on machine j at speed v, and 0 otherwise.

Table 2
Example for objectives calculation.

		$j=1$	$j=2$	$j=3$
$PP_{j,v}$	$v=1$	5	4	5
	$v=2$	20	16	20
SP_j		1	2	1
$t_{i,j}$	$i=1$	4	2	2
	$i=2$	2	2	2
	$i=3$	4	4	6
	$i=4$	4	4	5
	$i=5$	4	2	6
	$i=6$	3	6	4

3.3. Encoding and objective calculation

It is an important issue to decide how to represent a solution when designing effective algorithms for solving EEDPFSP. As described earlier in this article, the solution of the problem should include three parts, i.e., the assignment of jobs to factories, the job processing sequence in each factory, and processing speed of every job on each machine. The encoding scheme proposed by Wang and Wang [4] is used in this article. A set of F sequences, which is denoted as $\Pi = (\pi_1, \pi_2, \ldots, \pi_F)$, is used to represent the job processing sequence in each factory. Each one includes a string of numbers that represents a permutation of the jobs assigned to the factory. Let $V = (v_{ij})_{n\times m}$ denote the speed matrix, where $v_{i,j}$ is the processing speed of $O_{i,j}$. In this case, a solution can be represented as $sol = (\Pi, V) = (\pi_1, \pi_2, \ldots, \pi_F; V)$, where $\pi_f = (\pi_{f,1}, \pi_{f,2}, \ldots, \pi_{f,nf})$, $f = 1,2,\ldots, F$ is a sequence related to factory f, and n_f is the total number of jobs assigned to factory f.

Consider an example with two factories and six jobs. The two factories are identical and each has three machines with two different speeds. The standard processing times, the power consumption of each machine run at each speed and at stand-by mode are given in Table 2.

Consider a solution $\Pi = \{\pi_1, \pi_2; V\} = \{(5, 2, 1), (4, 3, 6); V\}$ and speed setting

$$V = \begin{bmatrix} 1 & 1 & 1 & 2 & 2 & 1 \\ 1 & 2 & 2 & 2 & 2 & 2 \\ 2 & 1 & 2 & 1 & 2 & 2 \end{bmatrix}^T$$

The Jobs 5, 2 and 1 are processed in order in factory 1. The real processing time of job 5 on machine 1 is $p_{5,1} = t_{5,1}/v_{5,1} = 4/2 = 2$. In the same way, we can get: $p_{5,2} = 1$, $p_{5,3} = 3$, $p_{2,1} = 2$, $p_{2,2} = 1$, $p_{2,3} = 2$, $p_{1,1} = 4$, $p_{1,2} = 2$, $p_{1,3} = 1$. The Gantt chart for the given solution ? is shown in Fig. 1. C(5) = 6, C(2) = 8, C(1) = 11. The total flow time of factory 1 is 25. $PEC_{1,1} = 2\times20+2\times5+4\times5 = 70$, $PEC_{1,2} = 1\times16+1\times16+2\times4 = 40$, $PEC_{1,3} = 3\times20+2\times5+1\times20 = 90$, $SEC_{1,1} = 0$, $SEC_{1,2} = 4\times2 = 8$, $SEC_{1,3} = 2\times1 = 2$. The total energy consumption of factory 1 is 210. In factory 2, C(4) = 9, C(3) = 12,C(6) = 14. The total flow time of factory 2 is 35. $PEC_{2,1} = 2\times20+4\times5+3\times5 = 75$, $PEC_{2,2} = 2\times16+2\times16+3\times16 = 112$, $PEC_{2,3} = 5\times5+3\times20+2\times20 = 120$, $SEC_{2,1} = 0$, $SEC_{2,2} = 3\times2 = 6$, $SEC_{2,3} = 0$. It can be concluded that the total energy consumption of factory 2 is 313. Finally, the total flow time and the energy consumption of the example instance is 60 and 523, respectively.

3.4. Operators

We designed the following operation operator.

(a) SpeedUp(f): In factory f, if there is idle time before a critical operation $O_{i,j}$, increase the speed of $O_{i,j-1}$ and then left-shift all the operations processed after $O_{i,j-1}$ correspondingly. The critical path in one factory of a certain schedule is shown in Fig. 1(a). The jobs and operations in the critical path are called critical jobs and critical operations, respectively. This operator can reduce the total flow time, but it may lead to an increase in the energy consumption. An example is shown in Fig. 2(a). The jobs with speed changed are shaded. Because there is idle time before the critical operation $O_{1,2}$, the total flow time can be reduced by speeding up the operation $O_{1,1}$.

(b) RandSpeedUp(f): In factory f, the processing speed of each job is randomly increased. Like the previous operator, this operator can reduce the total flow time, but it will increase the

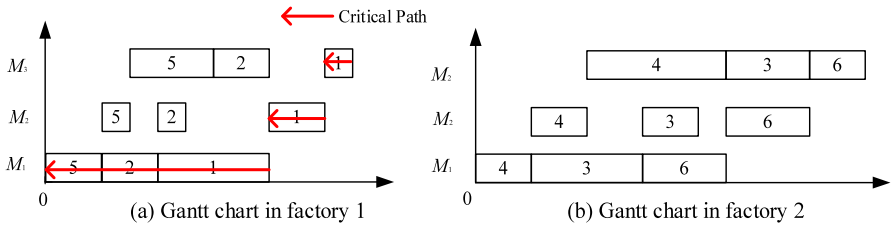

Fig. 1. A scheduling Gantt chart for the example instance.

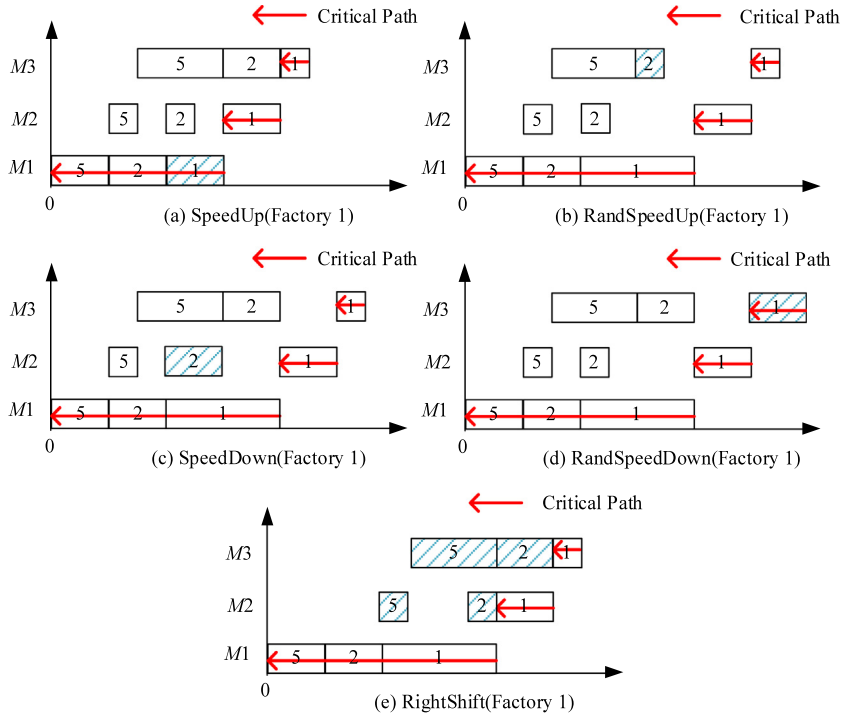

Fig. 2. Operators (Speedup, RandSpeedup, SpeedDown, RandSpeedDown, RightShift).

energy consumption. In Fig. 2(b), the speed of the operation $O_{2,3}$ is increased, which leads to a reduction in total flow time and an increase in energy consumption.

(c) SpeedDown(f): For the noncritical path in factory f, decrease the speeds of all the noncritical operations (except the operations on the last machine) as slow as possible without increasing total flow time. This operator can reduce the energy consumption while keeping the total flow time constant. In Fig. 2(c), the speed of $O_{2,2}$ is reduced, thereby reducing energy consumption.

(d) RandSpeepDown(f): In factory f, the processing speed of each job is randomly decreased. This operator can reduce the energy consumption, but it will increase the total flow time. An example is shown in Fig. 2(d) where the speed of $O_{1,3}$ is reduced.

(e) RightShift(f): In factory f , the jobs on the non-critical path are shifted to the right. The operator postpones the turn-on time of the machines, thereby reducing energy consumption, but will increase the total flow time. In Fig. 2(e), the jobs 5 and 1 are shifted to the right, thereby postponing the turn-on time of the machines 2 and 3.

As shown in Fig. 3(a), Job_1 and Job_2 are processed on the same machine. There is the idle time between Job_1 and Job_2 which is denoted as ΔT. The speed of Job_1 is v_1. There are two improvements proposals to reduce energy consumption. The first one is shown in Fig. 3(b), reducing the speed. The second is shown in Fig. 3(c), where Job_1 is shifted to the right while keeping the speed unchanged. According to the setting in the literature [4], $PP_{j,v}$=4 × v^{2}kW and $SP_j = 1$ kW.

The standard processing time of Job_1 is denoted as T_0, then the actual processing time $T_0/v_1 = T$. In Fig. 3(b), the speed of Job_1 is set to v_2 ($v_1 > v_2$). At the same time, it is assumed that processing with v_2 just fills up the idle time. That is, $T_0/v_2 = T+\Delta T$.

The energy consumption in Fig. 3(a)–(c) is calculated as follows.

$$E_a = PP_{j,v_1} \times T + SP_j \times \Delta T = 4 \times v_1^2 \times T + \Delta T$$
$$= 4 \times (\frac{T_0}{T})^2 \times T + \Delta T = 4 \times \frac{T_0^2}{T} + \Delta T.$$

$$E_b = PP_{j,v_2} \times (T + \Delta T) = 4 \times v_2^2 \times (T + \Delta T)$$
$$= 4 \times \left(\frac{T_0}{T + \Delta T}\right)^2 \times (T + \Delta T) = 4 \times \frac{T_0^2}{T + \Delta T}.$$

$$E_c = PP_{j,v_1} \times T = 4 \times v_1^2 \times T = 4 \times \left(\frac{T_0}{T}\right)^2 \times T = 4 \times \frac{T_0^2}{T}.$$

Then, $E_b < E_c < E_a$.

Finally, it is concluded that decreasing speed has priority over shifting right for reducing energy consumption.

All the above operators operate on the processing speed of every job on each machine. The following two operation operators are used to adjust the assignment of jobs to factories and the job processing sequence in each factory.

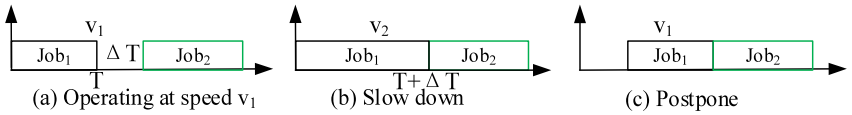

Fig. 3. Slow down and postpone.

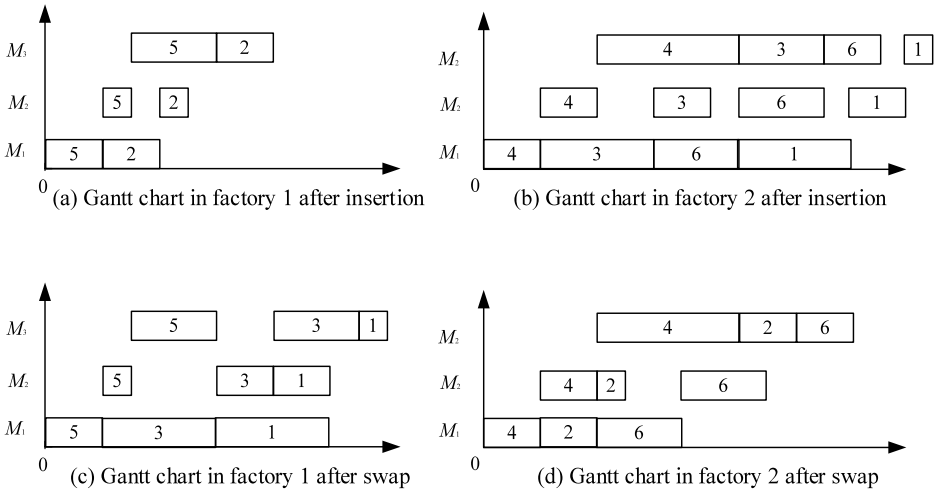

Fig. 4. Operators (Insert, Swap).

(f) Insert: This operator extracts a job from a factory, and then inserts it into a certain position of a factory. Please note that the factory for extracting job and the factory for inserting can be the same factory. Fig. 4(a) and (b) show the Gantt chart in factory 1 and 2 after the insertion which removes job 1 and insert it to factory 2.

(g) Swap: This operator selects a job and exchanges it with another job. The two jobs exchanged can be selected from the same factory or from two different factories. Fig. 4(c) and (d) show the Gantt chart in factory 1 and 2 after swapping jobs 2 and 3.

4. Improved NSGAII algorithm

4.1. Basic NSGAII

NSGAII is one of the most superior multi-objective evolutionary optimization algorithms [7]. Fig. 5 shows the pseudo-code of the basic NSGAII algorithm.

NSGAII was originally used to solve the optimization problem of continuous functions. Fewer pieces of literature were using NSGAII to solve the DPFSP. Therefore, besides the coding method and operators, it is necessary to design corresponding improvement strategies according to the characteristics of the DPFSP. These improvement strategies will be described in detail in the following sections.

Algorithm NSGAII

```
t = 1;
Initialize P(t) with PS solutions;
While (t < T) do
    Q(t) ← makenewpop(P(t));
    R(t) ← P(t) ∪ Q(t);
    F ← fastnondominatedsort R(t);//F = (F_1, F_1, ...)
    P(t + 1) ← ∅; i ← 1;
    repeat
        crowdingdistance(F_i);
        P(t + 1) ← P(t) ∪ F_i;
        i = i + 1;
    until(|P(t + 1)| + |F_i| ≤ N)
    sort(F_i, ≻_n);
    P(t + 1) ← P(t + 1) ∪ F_i[1: (N − |P(t + 1)|)];
    t ← t + 1;
endwhile
```

Fig. 5. The basic NSGAII.

4.2. Flowchart of the proposed algorithm

The flowchart of our proposed algorithm is illustrated in Fig. 6. We have made some improvements to the basic NSGAII. The improvement for the EEDPFSP is represented in the form of shading, which includes initialization, makenewpop, local intensification.

4.3. Initializing population

Our algorithm, INSGAII, starts from a population of *PS* initial solutions. The quality of the initial solutions has a direct impact on the performance of the whole algorithm. A good initial population will speed up the algorithm convergence. If there is an initial solution that minimizes one of the objectives in the population, the solution will tend to the optimal solution at a faster rate during evolution. For PFSP, the NEH heuristic algorithm [44] is the most effective algorithm, and has been widely used. The basic idea of NEH is that jobs with more total processing time (i.e., the sum of the processing time on all the machines) should be earlier processed. Firstly, a temporary job sequence is generated by sorting the total processing time of all the jobs in the descending order. Then, by continuously performing the insertion operation, a scheduling sequence that minimizes the value of a certain objective function is output. Based on the NEH algorithm, we propose 4 initial solution generation algorithms suitable for the characteristics of the problem in this paper.

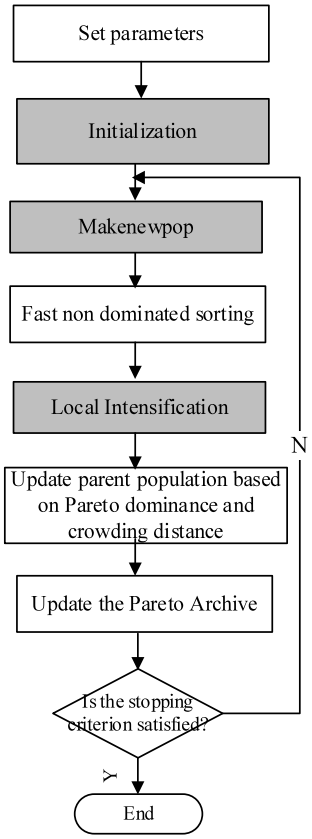

Fig. 6. Flowchart of the proposed algorithm.

The first heuristic algorithm is named MaxSMinTFT, whose pseudo-code is shown in Fig. 7. The general idea is to minimize the total flow time. First of all, all the jobs are processed at maximum speed to obtain the minimum processing time. Then, a job permutation is generated by sorting jobs according to the total actual processing time $T_i = \sum_j p_{i,j}$ in a descendant order. The jobs in the permutation are taken out one by one and inserted into the position of the factory that can lead to the minimum increase of total flow time. Then SpeedDown(f) operator is executed for each factory f to reduce the energy consumption while keeping the total flow time unchanged.

The second heuristic algorithm is named MinSMinTEC, which has the same frameworks as MaxSMinTFT except for the following different points. First, all the speeds are set to the minimum value. Then, when inserting jobs in factories, the goal is to minimize energy consumption. Finally, the operator RightShift(f) for each factory f is used to further reduce energy consumption. The pseudo-code for MinSMinTEC is shown in Fig. 8.

The third initialization method is named RandSMinTFT, where all the speeds of jobs are randomly generated. The other steps are the same as MaxSMinTFT. The fourth initialization method is named RandSMinTEC, in which all the speeds of jobs are randomly generated. The other steps are the same as MinSMinTEC.

The above heuristics generate 4 solutions. The remaining *PS*-4 solutions are randomly generated. The processing speed of each job on each machine is randomly generated. In addition, the jobs are randomly assigned to the factory for processing, and it is guaranteed that a factory has at least one job. In this way, we obtain an initial population with a high level of quality and diversity.

In addition, a Pareto Archives (PA) is used to record the non-dominant solutions for exploration and updated with the following rules. If a new solution is dominated by any solution in PA, it is not updated; otherwise, all the solutions dominated by the new solution in the PA are removed, and the new solution enters the PA. The pseudo-code of the update process is shown in Fig. 9. The newly generated PS solutions will be updated to PA.

4.4. Generate the next generation-makenewpop

Because the representation of the solution to our problem is different from the NSGAII source paper, we cannot directly use the method from NSGAII to generate the next generation. We need to design new generation methods for the next generation. We generate the next generation inspired by the discrete artificial bee colony algorithm [21]. We first design methods of generating a new individual, and then discuss how to generate the next generation.

4.4.1. Methods of generating a new individual

We design three individual generation methods, namely Insertion-based New-individual Generation Method (INGM), Swap-based New-individual Generation Method (SNGM) and

Algorithm MaxSMinTFT

$v_{i,j} = V_s, \forall i,j$; //Set all speeds to maximum;
$p_{i,j} = t_{i,j}/v_{i,j}, \forall i,j$; //Compute the actual process time for each job on each machine;
Compute $T_i = \sum_j p_{i,j}$ for each job $i \in N$;
Generate job permutation $\lambda=(\lambda_1,\lambda_2,...,\lambda_n)$ according to non-descending order of T_i;
$\pi:=(\pi_1,\pi_2,...,\pi_f)$,where $\pi_k = (\lambda_k), k = 1,2,\dots,f$;
Remove jobs $\lambda_1,\lambda_2,...,\lambda_f$ from λ;
While sizeof(λ) **do** //NEH enumeration procedure
    Extract the first job j from λ;
    for k:=1 to f **do**
        Test job j at all the possible positions $of\ \pi_k$;
        Δ_k := minimum increase of total flowtime;
        ξ_k := position resulting in Δ_k;
    endfor
    k^*: = arg $(\min_{k=1,2,\dots f} \Delta_k)$;
    Insert job j at position ξ_{k^*} of π_{k^*};
endwhile
SpeedDown(f), $\forall f$
return $sol = (\pi_1, \pi_2, \dots, \pi_F; V)$

Fig. 7. The MaxSMinTFT Heuristic algorithm.

Algorithm MaxSMinTEC

$v_{i,j} = V_1, \forall i,j$; //Set all speeds to minimum;
$p_{i,j} = t_{i,j}/v_{i,j}, \forall i,j$; //Compute the actual process time for each job on each machine;
Compute $T_i = \sum_j p_{i,j}$ for each job $i \in N$;
Generate job permutation λ=(λ_1,λ_2,...,λ_n) according to non-descending order of T_i;
π:=(π_1,π_2,...,π_f),where $\pi_k = (\lambda_k), k = 1,2,\dots,f$;
Remove jobs λ_1,λ_2,...,λ_f from λ;
While sizeof(λ) **do** //NEH enumeration procedure
    Extract the first job j from λ;
    for k:=1 to f **do**
        Test job j at all the possible positions $of\ \pi_k$;
        Δ_k := minimum increase of total energy consumption;
        ξ_k := position resulting in Δ_k;
    endfor
    k^*: = arg ($\min_{k=1,2,\dots f} \Delta_k$);
    Insert job j at position ξ_{k^*} of π_{k^*};
endwhile
RightShift(f), $\forall f$
return $sol = (\pi_1, \pi_2, \dots, \pi_F; V)$

Fig. 8. The MinSMinTEC Heuristic algorithm.

Algorithm UpdatetoArchive(*newSol*)

for each *solution* in PA **do**
    if *solution* $\succ$ *newSol* **do return**
endfor
for each *solution* in PA **do**
    if *newSol* $\succ$ *solution* **remove** *solution*
endfor
add *newSol* to PA

Fig. 9. The UpdatetoArchive procedure.

Hybrid-based New-individual Generation Method (HNGM), for generating new solutions in the neighborhood of an old solution.

(a) Insertion-based New-individual Generation Method (INGM)

The EEDPFSP in this article is a bi-objectives optimization problem. When a new individual is generated, an objective is randomly selected for optimization.

If the objective of the optimization is the total flow time, then select the factory with the largest total flow time (marked as f_{TFT}) for operation. Repeat the following process until a new solution dominates the current solution or half of the jobs in f_{TFT} have been tried. Randomly select a job in factory f_{TFT}, and then insert this job into all the positions in all the factories. Then the speeds of jobs in the two factories (f_{TFT} and the factory which the job inserted into) are randomly reduced, and then the speed of the jobs on the critical path of the two factories is increased. At this time, a new solution is obtained. If the new solution and the old solution do not dominate each other, update the newly obtained solution to PA.

If the total energy consumption is selected as the objective of optimization, then select the factory with the largest total energy consumption (marked as f_{TEC}) for operation. Repeat the following process until a new solution dominates the current solution or half of the jobs in f_{TEC} have been tried. Randomly select a job in factory f_{TEC}, and then insert this job into all the positions in all the factories. Then, three operators *RandSpeedDown*, *SpeedDown* and *RightShift* are used in the two factories (f_{TEC} and the factory which the job inserted into) to adjust the speeds of jobs to reduce energy consumption. And now, a new solution is obtained. If the new solution and the old solution do not dominate each other, update the newly obtained solution to PA.

The pseudo-code of INGM is shown in Fig. 10. In the pseudo-code, superscript 1 indicates the relevant parameters of sol^1. f^1_{TFT} denotes the factory with the largest total flow time in the solution sol^1. f^1_{TEC} is the factory with the largest total energy consumption in the solution sol^1. $p^1_{f_0}$ is the sequence of jobs in f_0.

(b) Swap-based New-individual Generation Method (SNGM)

SNGM and INGM are very similar. The difference is that the swap operation replaces the insert operation. The pseudo-code of SNGM is shown in Fig. 11.

(c) The Hybrid-based New-individual Generation method (HNGM)

The HNGM chooses the INGM with 50% probability, and the SNGM with another 50% probability.

4.4.2. The generation method of the next generation

Specifically, the method of employed bees and onlooker bees in the discrete artificial bee colony algorithm proposed by Pan [21] to generate the next generation *Q*. An employed bee randomly generates a neighboring solution for each solution in the population using the three methods described above: INGM, SNGM, and HNGM. There are two strategies for the onlooker bees. Firstly, the onlooker bees select solutions from the population *P* to further explore. Secondly, we choose a solution for an onlooker from the solutions generated by the employed bees using the binary tournament selection. After the selection, the onlookers utilize the INGM, SNGM, and HNGM to produce neighboring solutions.

The pseudo of the generation method of the next generation which is named makenewpop is shown in Fig. 12. In this pseudocode, the current population is *P*, and *sol(i)* is the *i*th individual of *P*. The final population of individuals generated is defined as *Q*.

It should be noted that the number of individuals in *Q* is 2**PS*. In our new algorithm, $R = Q$ instead of $R = P + Q$. Therefore, the number of individuals in *R* is 2**PS*.

4.5. Local intensification

As we all know, local intensification helps evolutionary algorithms to improve the performance of solving complex problems [4]. The local intensification is also used in our INSGAII algorithm. In our Local Intensification shown in Fig. 13, a job is extracted from the current solution sol^1 and reinserted to all

Algorithm INGM(sol)

sol^1=sol; $direction = rand()\%2$;
if $direction = 0$ **do** $f_0 = f^1_{TFT}$;
else $f_0 = f^1_{TEC}$; **endif**
$JobsTried = \emptyset$;
While sizeof($JobTried$) < sizeof($\pi^1_{f_0}$)/2 **do**
    Randomly extract one job j from ($\pi^1_{f_0}$-$JobTried$);
    Add job j to $JobTried$;
    if $direction == 0$ **do**
        RandSpeedUp(f_0); SpeedUp(f_0);
    else
        RandSpeedDown(f_0); SpeedDown(f_0); RightShift(f_0)
    ***e*ndif**
    for $f \coloneqq 1$ *to* F **do**
        for $l \coloneqq 1$ *to* t **do**
            Insert the job j to the position l of the factory f;
            if $direction == 0$ **do**
                RandSpeedUp(f); SpeedUp(f);
            else
                RandSpeedDown(f); SpeedDown(f); RightShift(f);
            ***e*ndif**
            if ($sol^1 \succ sol$)**do**
                return sol^1;
            ***el*seif**(!$sol \succ sol^1$)
                UpdatetoArchive(sol^1)
            ***e*ndif**
            Remove the job j from the position l of the factory f;
        ***e*ndfor**
    ***e*ndfor**
endwhile
return sol

Fig. 10. The INGM procedure.

the possible positions in any of the factories. Simultaneously, the speed is adjusted to get a better total flow time or total energy consumption. If the best solution among the neighboring solutions generated is better than the current solution, the current solution is replaced with the best neighboring solution. The process is repeated until all the jobs are considered and no improvement is found. In our INSGAII, we do not perform the local intensification to every solution found since it is very time-consuming. After sorting according to non-domination on the population R, the best non-dominated set F_1 is obtained. In each iteration, we perform the local intensification to the solution randomly selected from F_1.

5. Numerical results and comparisons

5.1. Experimental setting

In the experiments and comparisons, 600 instances are used to test the performance of our algorithm. Following [4,5], we set $F = \{2, 3, 4, 5\}$, $n = \{20, 40, 60, 80, 100\}$, and $m = \{4, 8, 16\}$. For each combination of $\{F, n, m\}$, there are ten instances. So, the number of instances is $4 \times 5 \times 3 \times 10 = 600$. The standard processing time $t_{i,j}$ is generated uniformly within range [5,50] and the processing speed v can be set as {1, 1.3, 1.55, 1.75, 2.10}. The Energy Consumption is set as $PP_{j,v} = 4 \times v^2$kW and $SP_j = 1$ kW.

The INSGAII is coded in C++ and all the instances are run on a PC with an Intel (R) Core (TM) i7-4790 CPU@3.60 GHz with 8.00 GB RAM in the Microsoft Windows 10 Operation System. In the following experiments, the termination criterion is set as $0.5 \times n$ s CPU time [4,5].

The EEDPFSP discussed in this article is a multi-objective optimization problem. The following four metrics [1] are used to evaluate the Pareto set obtained. To eradicate the dimension influence of metric values, a normalization method is employed in the following S-metric and D-metric.

(a) N-metric (Number of non-dominated solutions): It counts the number of the non-dominated solutions in the obtained Pareto archive $\boldsymbol{E}$, denoted as $|\boldsymbol{E}|$.

(b) C-metric (Set coverage): It measures the dominance relationship between solutions in two Pareto archives $\boldsymbol{E}_1$ and $\boldsymbol{E}_2$. $C(\boldsymbol{E}_1, \boldsymbol{E}_2)$ is calculated as follows, which reflects the percentage of the solutions in $\boldsymbol{E}_2$ that are dominated by or the same as the solutions in $\boldsymbol{E}_1$:

$$C(E_1, E_1) = |\{b \in E_2 | \exists a \in E_1, a \succ b \text{ or } a = b\}|/|E_2| \quad (18)$$

(c) S-metric (Spread): This metric is a distribution indicator, which can measure the diversity and distribution quality of the

Algorithm SNGM(sol)

sol^1=sol; $direction = rand()\%2$;
if $direction = 0$ **do** $f_0 = f^1_{TFT}$;
else $f_0 = f^1_{TEC}$; **endif**
$JobsTried = \emptyset$;
While sizeof($JobTried$) < sizeof($\pi^1_{f_0}$)/2 **do**
  Randomly select one job j from ($\pi^1_{f_0}$-$JobTried$);
  Add job j to $JobTried$;
  for $f \coloneqq 1$ *to* F **do**
    for $l \coloneqq 1$ *to* t **do**
      Swap the job j and the job k in the position l of the factory f;
      if $direction == 0$ **do**
        RandSpeedUp(f_0); SpeedUp(f_0); RandSpeedUp(f); SpeedUp(f);
      else
        RandSpeedDown(f_0); SpeedDown(f_0); RightShift(f_0);
        RandSpeedDown(f); SpeedDown(f); RightShift(f);
      ***e*ndif**
      if ($sol^1 \succ sol$)**do**
        return sol^1;
      ***e*lseif**(!$sol \succ sol^1$)
        UpdatetoArchive(sol^1)
      ***e*ndif**
      Swap the two jobs (j,k) and restore their original positions;
    ***e*ndfor**
  ***e*ndfor**
endwhile
return sol

Fig. 11. The SNGM procedure.

Algorithm makenewpop(PS, Ξ, ω)

for $i \coloneqq 1$ to PS **do** //employed bees phase
  $sol'(i)$:=neighbor of $sol(i)$ generated by operator Ξ;
  Put $sol'(i)$ into X';
endfor
for $i \coloneqq 1$ to PS **do** //onlooker bees phase
  if $\omega == 0$ **do**
    sol':=solution selected from P using tourmonent selection;
  else
    sol':=solution selected from X' using tourmonent selection;
  endif
  $sol''(i)$:=neighbor of sol' generated by operator Ξ ;
  Put $sol''(i)$ into X'';
endfor
$Q = X' \cup X''$;
return Q

Fig. 12. The makenewpop procedure.

solution set. It can be defined as:

$$S(\mathrm{A}, PF^*) = \frac{\sum_{j=1}^{m} d_j^e + \sum_{i=1}^{|A|} |d_i - \overline{d}|}{\sum_{j=1}^{m} d_j^e + |A| \cdot \overline{d}} \tag{19}$$

In the (19), A is the Pareto Front obtain by an algorithm, PF* is the true PF. d_i is the Euclidean distance between the ith solution and its closest solution in A, $\overline{d}$ is the mean of all the d_i, |A| is the number of members in A, and m is the number of objectives, d_j^e indicates the Euclidean distance between the extreme solution in terms of the jth objective in A and the corresponding extreme solution in PF*. For this metric, a lower value is expected. A smaller value indicates a more uniform distribution along the PF. Since the true PF is unknown, in the actual calculations, PF* is the non-dominated subset of all non-dominated solutions found by all algorithms.

(d). D-metric (Inverse Generational Distance): This indicator is a comprehensive metric that can measure the diversity and convergence of the solution set, which can be obtained by:

$$D(\mathrm{A}, PF^*) = \frac{\sum_{v \in PF^*}^{d} (v, A)}{|PF^*|} \tag{20}$$

Algorithm LocalIntensification(sol)

sol^1=sol; $direction = rand()\%2$;
Put the jobs of each factory into a sequence $Permutation$ in turn;
$Cnt = 0; k = 0; Update = false$;
While $Cnt < n$ **do**
  f^1 is the factory the job $Permutation[k]$ processed in;
  Extract the job $Permutation[k]$ from f^1;
  if $direction == 0$ **do**
    RandSpeedUp(f^1); SpeedUp(f^1);
  else
    RandSpeedDown(f^1); SpeedDown(f^1); RightShift(f^1)
  ***e*ndif**
  for $f \coloneqq 1$ *to* F **do**
    for $l \coloneqq 1$ *to* t **do**
      Insert the job $Permutation[k]$ to the position l of the factory f;
      if $direction == 0$ **do**
        RandSpeedUp(f); SpeedUp(f);
      else
        RandSpeedDown(f); SpeedDown(f); RightShift(f);
      ***e*ndif**
      if $(sol^1 \succ sol)$**do**
        $sol = sol^1$;$Updated$=true;
      ***e*lse**
        if(!$sol \succ sol^1$) **do** UpdatetoArchive(sol^1); **endif**
        $sol^1 = sol$;
      ***e*ndif**
      Remove the job $Permutation[k]$ from the position l of the factory f;
    ***e*ndfor**
  ***e*ndfor**
  if $(Updated)$**do** $Cnt = 0$; **else** $Cnt++$; **endif**
  $k \coloneqq (k+1)\%n + 1$;
endwhile

Fig. 13. The local intensification procedure.

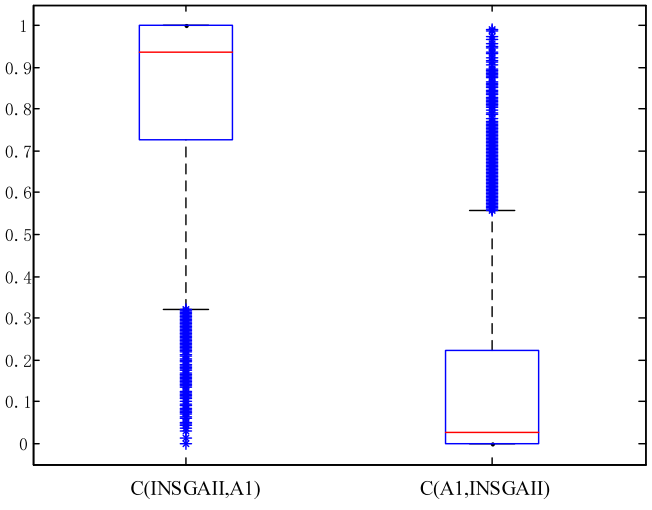

Fig. 14. Boxplot of C metric with or without four heuristics initialization.

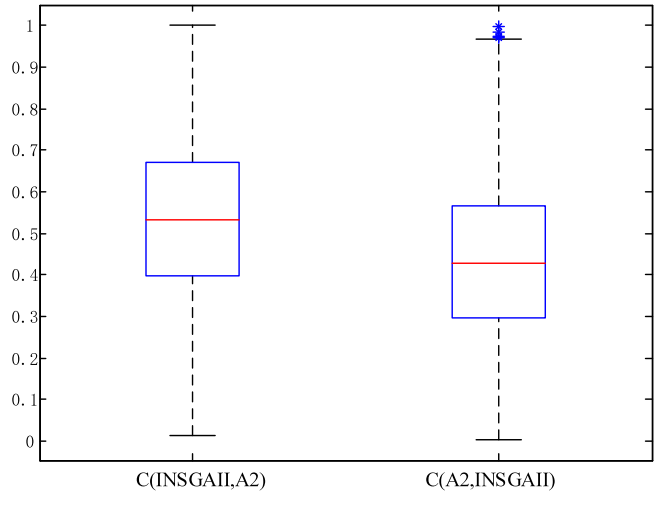

Fig. 15. Boxplot of C metric with or without local intensification.

In the (20), $d(v, A)$ is the minimum Euclidean distance between v and the points in A, and $|PF^*|$ is the number of points in PF^*.

5.2. Parameter setting

The INSGAII contains three key parameters: (1) population size (PS); (2) Methods of generating a new individual (Ξ); and (3) the operator type for onlooker bees (ω). For these three parameters, we first determine the general scope of them based on existing literature. Then, we conduct a preliminary experiment to determine the levels for each parameter. The levels for these three parameters are: PS at three levels: 10, 20 and 30; Ξ at three levels: INGM, SNGM and HNGM; ω at two levels: 0 and 1.

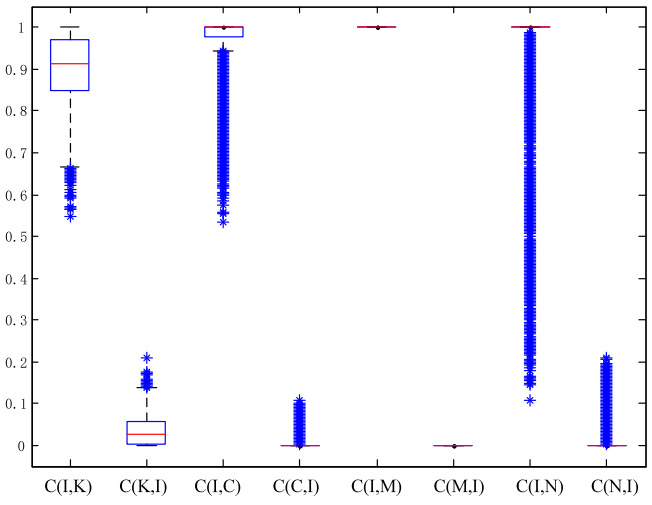

Fig. 16. Boxplot of C metric for different pair (I denotes INSGAII, K denotes KCA, C denotes CMA, M denotes MOEA/D, N denotes NSGAII).

Table 3
Parameter array and RV values.

PS	Ξ	ω	RV (%)
30	*INGM*	0	16.80
30	*SNGM*	0	16.65
30	*HNGM*	0	16.28
20	*INGM*	0	16.20
30	*INGM*	1	16.18
20	*INGM*	1	16.06
20	*HNGM*	0	16.04
20	*SNGM*	0	16.02
10	*INGM*	0	16.00
30	*HNGM*	1	15.97
30	*SNGM*	1	15.95
10	*HNGM*	0	15.91
10	*SNGM*	0	15.89
20	*HNGM*	1	15.89
10	*HNGM*	1	15.89
10	*INGM*	1	15.86
20	*SNGM*	1	15.81
10	*SNGM*	1	15.73

The above factors result in a total of $3 \times 3 \times 2 = 18$ different configurations for the INSGAII algorithm.

A total of 60 instances with different scales (F, n, m) are used for investigation. For every instance, the INSGAII with each (PS, Ξ, ω) is run ten times independently to obtain Pareto archives $\boldsymbol{E_i}$ ($i = 1, 2, \ldots, 18$). The rigid non-dominated solutions among $\boldsymbol{E_1}$–$\boldsymbol{E_{18}}$ consist of the final set $\boldsymbol{FE}$. Then, the contribution of $\boldsymbol{E_i}$ is calculated as $\mathrm{CON}(i) = |E_i'|/|\boldsymbol{FE}|$, where $E_i' = \boldsymbol{E_i} \cap \boldsymbol{FE}$. After testing all the instances, it calculates the average CON of each parameter combination as the response variable value (RV). The parameter array and the RV values are listed in Table 3.

From Table 3, it can be seen that when $PS = 30$, $\Xi =$ INGM and $\omega = 0$, the algorithm gets the best result. Therefore, parameters are set as $PS = 30$, $\Xi =$ INGM and $\omega = 0$ for the following tests and comparisons.

5.3. Effect of initialization

To demonstrate the effectiveness of the initialization with four heuristics, we compare the INSGAII to the INSGAII with random initialization (denoted as A1). For each scale (F, n, m), the average C metric of ten instances (each instance is run ten times) is summarized in Table 4 and the boxplots of all C metrics are shown in Fig. 14. We use the nonparametric Kruskal–Wallis test

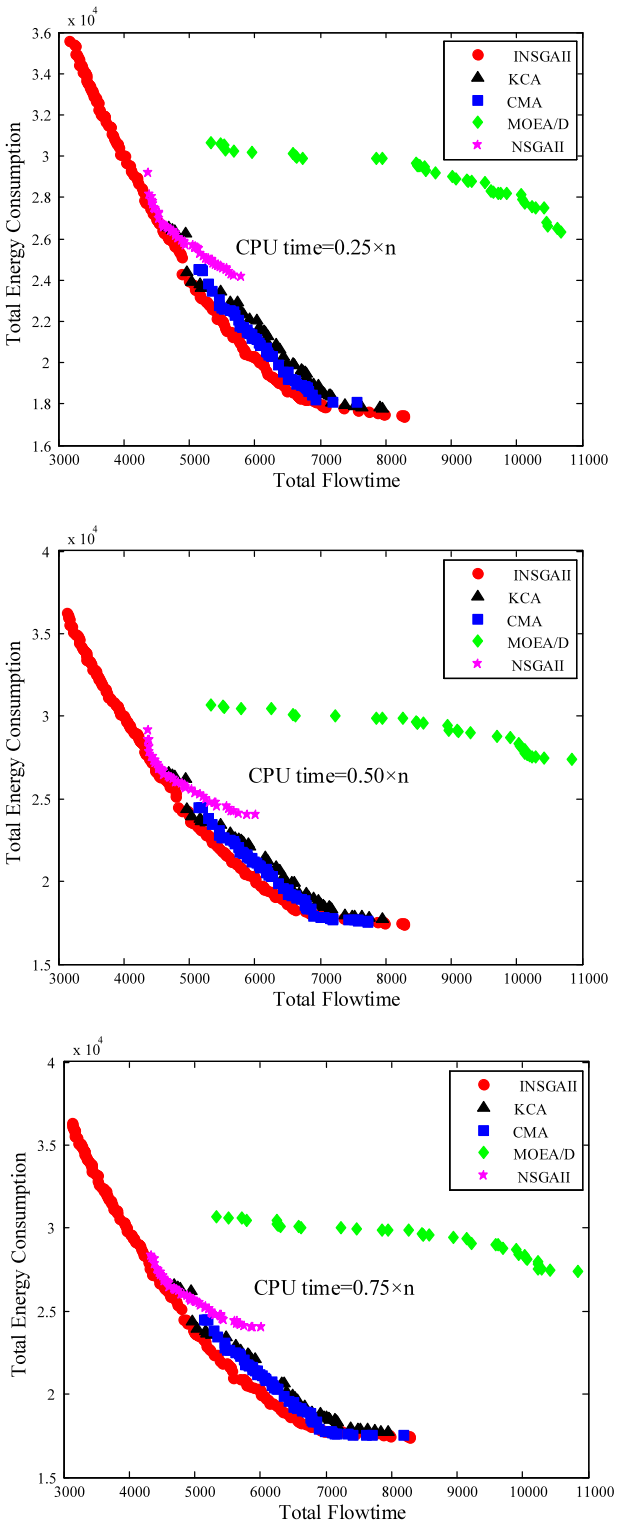

Fig. 17. Pareto fronts by algorithms with different stopping criteria.

with 95% confidence to analyze whether the difference between the INSGAII and A1 is significant or not. The p values are listed in Table 4.

From Table 4, it can be seen that C(INSGAII, A1) is much larger than C(A1, INSGAII) on all the instances, which implies that the most non-dominated solutions obtained by A1 are dominated by those obtained by the INSGAII. In addition, almost all p values are

Table 4
Comparisons between INSGAII and A1 (I denotes INSGAII).

(n,m)	C(I,A1)	C(A1, I)	p	C(I,A1)	C(A1, I)	p	C(I,A1)	C(A1, I)	p	C(I,A1)	C(A1, I)	p
	F = 2			F = 3			F = 4			F = 5		
(20,4)	0.536	0.412	0	0.507	0.430	0.01	0.529	0.424	0	0.482	0.456	0.337
(20,8)	0.557	0.384	0	0.536	0.405	0	0.526	0.412	0	0.538	0.399	0
(20,16)	0.608	0.349	0	0.570	0.380	0	0.563	0.383	0	0.496	0.436	0.034
(40,4)	0.719	0.243	0	0.682	0.276	0	0.684	0.273	0	0.711	0.248	0
(40,8)	0.887	0.093	0	0.823	0.150	0	0.843	0.122	0	0.782	0.179	0
(40,16)	0.940	0.042	0	0.928	0.052	0	0.921	0.059	0	0.895	0.083	0
(60,4)	0.802	0.175	0	0.855	0.113	0	0.841	0.127	0	0.880	0.088	0
(60,8)	0.915	0.054	0	0.939	0.043	0	0.941	0.037	0	0.931	0.052	0
(60,16)	0.965	0.021	0	0.962	0.021	0	0.949	0.030	0	0.961	0.020	0
(80,4)	0.919	0.062	0	0.907	0.071	0	0.861	0.103	0	0.931	0.046	0
(80,8)	0.957	0.025	0	0.965	0.020	0	0.954	0.023	0	0.953	0.028	0
(80,16)	0.988	0.005	0	0.975	0.010	0	0.980	0.006	0	0.980	0.006	0
(100,4)	0.923	0.057	0	0.901	0.071	0	0.926	0.047	0	0.912	0.057	0
(100,8)	0.985	0.005	0	0.975	0.010	0	0.961	0.013	0	0.963	0.012	0
(100,16)	0.997	0.002	0	0.989	0.005	0	0.982	0.004	0	0.980	0.005	0

Table 5
Comparisons between INSGAII and A2 (I denotes INSGAII).

(n,m)	C(I,A2)	C(A2, I)	p	C(I,A2)	C(A2, I)	p	C(I,A2)	C(A2, I)	p	C(I,A2)	C(A2, I)	p
	F = 2			F = 3			F = 4			F = 5		
(20,4)	0.553	0.398	0	0.524	0.425	0	0.520	0.442	0	0.524	0.434	0.002
(20,8)	0.556	0.399	0	0.531	0.428	0	0.561	0.402	0	0.600	0.358	0
(20,16)	0.516	0.446	0	0.505	0.463	0.043	0.522	0.440	0.001	0.506	0.459	0.069
(40,4)	0.559	0.401	0	0.511	0.461	0.108	0.534	0.424	0.001	0.566	0.397	0
(40,8)	0.532	0.431	0.001	0.494	0.475	0.801	0.546	0.424	0	0.508	0.462	0.113
(40,16)	0.514	0.463	0.035	0.512	0.469	0.133	0.528	0.451	0.007	0.499	0.472	0.234
(60,4)	0.537	0.427	0.002	0.556	0.409	0	0.486	0.477	0.914	0.507	0.450	0.119
(60,8)	0.507	0.472	0.238	0.503	0.472	0.181	0.518	0.456	0.036	0.507	0.469	0.225
(60,16)	0.491	0.472	0.355	0.496	0.493	0.949	0.489	0.503	0.424	0.537	0.448	0.001
(80,4)	0.622	0.339	0	0.572	0.394	0	0.553	0.407	0	0.561	0.396	0
(80,8)	0.522	0.451	0.005	0.524	0.446	0.005	0.504	0.469	0.149	0.529	0.439	0.002
(80,16)	0.507	0.443	0.016	0.516	0.461	0.016	0.541	0.450	0.001	0.515	0.469	0.103
(100,4)	0.626	0.328	0	0.554	0.404	0	0.551	0.403	0	0.540	0.424	0
(100,8)	0.575	0.375	0	0.536	0.425	0	0.558	0.410	0	0.535	0.427	0
(100,16)	0.589	0.355	0	0.536	0.438	0	0.526	0.447	0	0.531	0.451	0.001

Table 6
C-metric of the algorithms ($F = 2$, I denotes INSGAII, K denotes KCA, C denotes CMA, M denotes MOEA/D, N denotes NSGAII).

(n,m)	C(I,K)	C(K,I)	p	C(I,C)	C(C,I)	p	C(I,M)	C(M,I)	p	C(I,N)	C(N,I)	p
(20,4)	0.980	0.004	0	0.781	0.045	0	1.000	0.000	0	0.974	0.005	0
(20,8)	0.981	0.005	0	0.864	0.031	0	1.000	0.000	0	0.986	0.002	0
(20,16)	0.977	0.006	0	0.988	0.003	0	1.000	0.000	0	0.913	0.011	0
(40,4)	0.950	0.013	0	0.952	0.003	0	1.000	0.000	0	1.000	0.000	0
(40,8)	0.910	0.033	0	0.999	0.000	0	1.000	0.000	0	1.000	0.000	0
(40,16)	0.891	0.047	0	0.997	0.000	0	1.000	0.000	0	1.000	0.000	0
(60,4)	0.910	0.035	0	1.000	0.000	0	1.000	0.000	0	0.999	0.000	0
(60,8)	0.854	0.053	0	1.000	0.000	0	1.000	0.000	0	1.000	0.000	0
(60,16)	0.844	0.075	0	0.985	0.001	0	1.000	0.000	0	1.000	0.000	0
(80,4)	0.896	0.040	0	1.000	0.000	0	1.000	0.000	0	1.000	0.000	0
(80,8)	0.859	0.054	0	0.998	0.000	0	1.000	0.000	0	1.000	0.000	0
(80,16)	0.827	0.077	0	0.998	0.000	0	1.000	0.000	0	1.000	0.000	0
(100,4)	0.881	0.046	0	0.998	0.000	0	1.000	0.000	0	1.000	0.000	0
(100,8)	0.852	0.056	0	1.000	0.000	0	1.000	0.000	0	1.000	0.000	0
(100,16)	0.788	0.083	0	0.987	0.001	0	1.000	0.000	0	1.000	0.000	0

less than 0.05, which means that the INSGAII is significantly better than A1. Therefore, the initialization with the four proposed heuristics is effective for solving the problem.

5.4. Effect of local intensification

For further verification about C metric, we compare the INSGAII to the INSGAII without local intensification (denoted as A2). For each scale (F, n, m), the average C metric of ten instances (each instance is run ten times) is summarized in Table 5. The nonparametric Kruskal–Wallis test with 95% confidence level is also tested and all p values are given in Table 5.

From Table 5, it can be seen that C(INSGAII, A2) is larger than C(A2, INSGAII) in most instances. For most instances, the difference is significant since $p < 0.05$. Fig. 15 shows that the INSGAII is better than A2 on average. Therefore, local intensification is effective in solving the problem.

5.5. Comparisons to other algorithms

Next, we compare the INSGAII to the existing algorithms for solving the EEDPFSP, including the KCA [4], the CMA [5], MOEA/D [6] and NSGAII [7]. By modifying the optimization objective function, we re-implement the KCA and CMA to suit our problem. In addition, we follow all the original details given

Table 7
N-metric, *S*-metric, *D*-metric values of the algorithms ($F = 2$, I denotes INSGAII, K denotes KCA, C denotes CMA, M denotes MOEA/D, N denotes NSGAII).

(*n*,*m*)	*N*-metric					*S*-metric					*D*-metric				
	I	K	C	M	N	I	K	C	M	N	I	K	C	M	N
(20,4)	**193.73**	46.8	60.01	40.56	50.14	**0.246**	0.498	0.521	0.561	0.549	**0.000**	0.012	0.014	0.041	0.016
(20,8)	**216.39**	52.99	65.88	41.36	44.52	**0.248**	0.555	0.539	0.605	0.601	**0.001**	0.019	0.036	0.063	0.042
(20,16)	**239.4**	53.89	60.81	40.01	42.77	**0.236**	0.595	0.571	0.605	0.621	**0.001**	0.029	0.086	0.114	0.092
(40,4)	**273.04**	57.59	63.38	47.39	46.46	**0.305**	0.465	0.518	0.656	0.588	**0.001**	0.035	0.038	0.102	0.043
(40,8)	**275.52**	61.52	62.64	47.41	40.15	**0.267**	0.541	0.575	0.654	0.619	**0.002**	0.056	0.082	0.149	0.099
(40,16)	**287.06**	66.39	40.1	43.64	37	**0.319**	0.577	0.625	0.637	0.634	**0.003**	0.064	0.185	0.255	0.212
(60,4)	**326.86**	69.29	62.18	47.45	44.4	**0.339**	0.464	0.533	0.655	0.596	**0.002**	0.061	0.063	0.169	0.076
(60,8)	**321.54**	70.21	46.86	43.98	38.39	**0.265**	0.512	0.599	0.640	0.623	**0.004**	0.105	0.135	0.226	0.165
(60,16)	**314.49**	74.36	29.27	33.93	33.81	**0.396**	0.557	0.642	0.633	0.644	**0.007**	0.117	0.319	0.398	0.346
(80,4)	**390.08**	78.9	52.8	45.77	44.85	**0.372**	0.460	0.558	0.604	0.605	**0.003**	0.086	0.094	0.210	0.108
(80,8)	**377.68**	80.34	37.87	41.27	36.83	**0.312**	0.515	0.622	0.632	0.628	**0.005**	0.151	0.209	0.318	0.235
(80,16)	**342**	84.68	25.57	32.23	32.9	**0.420**	0.562	0.657	0.633	0.646	**0.012**	0.163	0.454	0.543	0.476
(100,4)	**418.17**	91.02	47.32	43.56	43.27	**0.392**	0.474	0.596	0.586	0.617	**0.004**	0.118	0.139	0.271	0.154
(100,8)	**393.38**	89.13	31.11	33.83	37.04	**0.344**	0.499	0.638	0.611	0.631	**0.008**	0.203	0.287	0.406	0.305
(100,16)	**326.14**	95.72	21.12	32.5	31.02	**0.424**	0.564	0.662	0.635	0.649	**0.020**	0.233	0.602	0.692	0.610

Table 8
C-metric of the algorithms ($F = 3$, I denotes INSGAII, K denotes KCA, C denotes CMA, M denotes MOEA/D, N denotes NSGAII).

(*n*,*m*)	*C*(I,K)	*C*(K,I)	*p*	*C*(I,C)	*C*(C,I)	*p*	*C*(I,M)	*C*(M,I)	*p*	*C*(I,N)	*C*(N,I)	*p*
(20,4)	0.991	0.002	0	0.761	0.045	0	1.000	0.000	0	0.906	0.020	0
(20,8)	0.994	0.001	0	0.838	0.032	0	1.000	0.000	0	0.802	0.033	0
(20,16)	0.986	0.002	0	0.976	0.006	0	1.000	0.000	0	0.634	0.056	0
(40,4)	0.957	0.011	0	0.875	0.008	0	1.000	0.000	0	0.999	0.000	0
(40,8)	0.942	0.018	0	1.000	0.000	0	1.000	0.000	0	1.000	0.000	0
(40,16)	0.948	0.014	0	0.999	0.000	0	1.000	0.000	0	1.000	0.000	0
(60,4)	0.916	0.023	0	0.999	0.000	0	1.000	0.000	0	1.000	0.000	0
(60,8)	0.886	0.039	0	1.000	0.000	0	1.000	0.000	0	1.000	0.000	0
(60,16)	0.847	0.054	0	0.999	0.000	0	1.000	0.000	0	1.000	0.000	0
(80,4)	0.910	0.036	0	1.000	0.000	0	1.000	0.000	0	1.000	0.000	0
(80,8)	0.871	0.057	0	1.000	0.000	0	1.000	0.000	0	1.000	0.000	0
(80,16)	0.833	0.069	0	1.000	0.000	0	1.000	0.000	0	1.000	0.000	0
(100,4)	0.889	0.048	0	1.000	0.000	0	1.000	0.000	0	1.000	0.000	0
(100,8)	0.860	0.061	0	1.000	0.000	0	1.000	0.000	0	1.000	0.000	0
(100,16)	0.796	0.083	0	1.000	0.000	0	1.000	0.000	0	1.000	0.000	0

Table 9
N-metric, *S*-metric, *D*-metric values of the algorithms ($F = 3$, I denotes INSGAII, K denotes KCA, C denotes CMA, M denotes MOEA/D, N denotes NSGAII).

(*n*,*m*)	*N*-metric					*S*-metric					*D*-metric				
	I	K	C	M	N	I	K	C	M	N	I	K	C	M	N
(20,4)	**212.26**	53.71	60.57	51.16	58.37	**0.207**	0.522	0.528	0.598	0.564	**0.000**	0.012	0.015	0.039	0.015
(20,8)	**231.83**	59.39	68.99	54.78	54.3	**0.236**	0.585	0.542	0.622	0.600	**0.000**	0.016	0.036	0.058	0.037
(20,16)	**269.22**	63.12	68.9	57.13	54.5	**0.254**	0.585	0.580	0.608	0.621	**0.001**	0.028	0.087	0.105	0.086
(40,4)	**283.26**	63.58	66.59	67.41	53.98	**0.353**	0.478	0.544	0.676	0.588	**0.001**	0.031	0.035	0.083	0.039
(40,8)	**301.28**	66.85	65.84	64.63	47.95	**0.345**	0.576	0.574	0.663	0.623	**0.002**	0.043	0.077	0.139	0.094
(40,16)	**325.17**	69.41	48.76	63.03	47	**0.300**	0.579	0.621	0.631	0.636	**0.003**	0.051	0.177	0.233	0.194
(60,4)	**366.7**	74.27	64.03	71.87	52.43	**0.280**	0.471	0.530	0.683	0.607	**0.001**	0.054	0.059	0.140	0.069
(60,8)	**352.15**	76.3	54.67	69.1	44.59	**0.357**	0.538	0.596	0.672	0.626	**0.003**	0.091	0.124	0.215	0.155
(60,16)	**366.88**	80.38	35.01	64.75	43.07	**0.386**	0.592	0.644	0.647	0.643	**0.006**	0.102	0.305	0.381	0.330
(80,4)	**433.58**	85.26	61.43	75.27	51.16	**0.368**	0.473	0.556	0.684	0.612	**0.002**	0.079	0.086	0.212	0.102
(80,8)	**418.11**	87.92	44.48	73.31	42.88	**0.337**	0.522	0.612	0.671	0.630	**0.004**	0.137	0.196	0.307	0.225
(80,16)	**403.94**	90.49	30.26	55.28	40.18	**0.361**	0.584	0.656	0.642	0.650	**0.011**	0.148	0.443	0.527	0.470
(100,4)	**476.37**	98	51.9	75.91	51.48	**0.332**	0.468	0.574	0.677	0.616	**0.003**	0.107	0.123	0.272	0.141
(100,8)	**463.55**	98.54	35.6	72.18	42.7	**0.261**	0.518	0.633	0.674	0.638	**0.006**	0.190	0.268	0.391	0.291
(100,16)	**421.59**	99.99	24.08	50.79	36.71	**0.387**	0.588	0.661	0.643	0.654	**0.016**	0.203	0.576	0.673	0.603

in each paper to achieve the performance it deserves. For the MOEA/D and the NSGAII, the next generation individual generation method of the original paper is no longer applicable, and a new next generation individual generation method is designed. The MOEA/D algorithm randomly selects a job and inserts it at a random location, and the speed is adjusted randomly. The NSGAII selects a job in the factory f_{TEC} or the factory f_{TFT}, inserts the job in all positions, and adjusts the speed randomly, and replaces the current solution if the new solution dominates the current solution. Furthermore, we also use the method in Section 5 to adjust the parameter values of the compared algorithm. Finally, the parameters of all the comparison algorithms are set as follows: in the KCA, $PS = 30$, $LS = 100$, and $PE = 60$; in the CMA the PS is 30 and times of local search is *100*; in the MOEA/D, $PS = 30$, $T = 10$; in the NSGAII, $PS = 30$. All the algorithms use $0.5 \times n$ *s* CPU time as the stopping criterion.

For an overall comparison, the average *C* metric, *N*-metric, *S*-metric and *D*-metric in solving all the instances are listed in Tables 6–13 grouped by a different number of *F*. The best value for each (*n*, *m*) pair is marked in bold. The boxplots of all the *C* metric are shown in Fig. 16. In addition, the non-parametric Kruskal–Wallis test is performed between the INSGAII and the KCA (CMA, MOEA/D and NSGAII) to show whether the difference is significant or not. The resulted *p* values are listed in the tables.

First of all, it can be seen that $C(\text{INSGAII, KCA}) =/\approx 1$ and $C(\text{KCA, INSGAII}) =/\approx 0$ almost on all (*n*, *m*) pairs, which means

Table 10
C-metric of the algorithms ($F = 4$, I denotes INSGAII, K denotes KCA, C denotes CMA, M denotes MOEA/D, N denotes NSGAII).

(*n*,*m*)	*C*(I,K)	*C*(K,I)	*p*	*C*(I,C)	*C*(C,I)	*p*	*C*(I,M)	*C*(M,I)	*p*	*C*(I,N)	*C*(N,I)	*p*
(20,4)	0.986	0.002	0	0.763	0.042	0	1.000	0.000	0	0.775	0.053	0
(20,8)	0.990	0.002	0	0.837	0.029	0	1.000	0.000	0	0.567	0.086	0
(20,16)	0.989	0.002	0	0.981	0.004	0	1.000	0.000	0	0.407	0.098	0
(40,4)	0.943	0.010	0	0.807	0.010	0	1.000	0.000	0	0.997	0.000	0
(40,8)	0.931	0.013	0	0.983	0.001	0	1.000	0.000	0	0.996	0.000	0
(40,16)	0.931	0.017	0	0.999	0.000	0	1.000	0.000	0	0.987	0.002	0
(60,4)	0.917	0.023	0	0.976	0.000	0	1.000	0.000	0	1.000	0.000	0
(60,8)	0.902	0.030	0	1.000	0.000	0	1.000	0.000	0	1.000	0.000	0
(60,16)	0.871	0.035	0	1.000	0.000	0	1.000	0.000	0	1.000	0.000	0
(80,4)	0.897	0.039	0	1.000	0.000	0	1.000	0.000	0	1.000	0.000	0
(80,8)	0.866	0.059	0	1.000	0.000	0	1.000	0.000	0	1.000	0.000	0
(80,16)	0.818	0.060	0	1.000	0.000	0	1.000	0.000	0	1.000	0.000	0
(100,4)	0.893	0.042	0	1.000	0.000	0	1.000	0.000	0	1.000	0.000	0
(100,8)	0.842	0.063	0	1.000	0.000	0	1.000	0.000	0	1.000	0.000	0
(100,16)	0.779	0.079	0	1.000	0.000	0	1.000	0.000	0	1.000	0.000	0

Table 11
N-metric, *S*-metric, *D*-metric values of the algorithms ($F = 4$, I denotes INSGAII, K denotes KCA, C denotes CMA, M denotes MOEA/D, N denotes NSGAII).

(*n*,*m*)	*N*-metric					*S*-metric					*D*-metric				
	I	K	C	M	N	I	K	C	M	N	I	K	C	M	N
(20,4)	**225.37**	61.75	65.11	60.99	62.86	**0.292**	0.553	0.538	0.604	0.559	**0.000**	0.012	0.015	0.036	0.013
(20,8)	**255.12**	65.33	73.16	64.77	61.05	**0.251**	0.605	0.550	0.613	0.599	**0.001**	0.018	0.038	0.058	0.037
(20,16)	**292.05**	67.62	70.82	68.97	61.3	**0.249**	0.611	0.581	0.599	0.620	**0.001**	0.029	0.092	0.097	0.080
(40,4)	**309.63**	67.56	68.6	75.59	60.21	**0.310**	0.494	0.553	0.677	0.596	**0.001**	0.032	0.037	0.081	0.040
(40,8)	**324.14**	69.68	70.24	85.82	54.47	**0.311**	0.583	0.572	0.667	0.620	**0.002**	0.044	0.075	0.136	0.090
(40,16)	**356.61**	71.78	52.6	81.89	55.07	**0.349**	0.601	0.620	0.628	0.638	**0.003**	0.059	0.182	0.230	0.193
(60,4)	**382.46**	79.77	74.2	90.9	58.5	**0.295**	0.479	0.546	0.690	0.605	**0.001**	0.049	0.056	0.132	0.067
(60,8)	**382.71**	83.27	60.18	92.35	51.12	**0.335**	0.547	0.595	0.676	0.631	**0.003**	0.087	0.127	0.210	0.152
(60,16)	**403.68**	81.34	39.1	87.03	49.5	**0.370**	0.600	0.645	0.643	0.644	**0.006**	0.099	0.301	0.368	0.316
(80,4)	**453.09**	90.22	62.99	95.61	58.47	**0.337**	0.480	0.558	0.687	0.606	**0.002**	0.073	0.081	0.182	0.097
(80,8)	**448.38**	92.61	47.18	94.12	50.48	**0.305**	0.546	0.617	0.677	0.635	**0.005**	0.128	0.186	0.294	0.220
(80,16)	**448.34**	90.37	32.48	86.54	47.22	**0.365**	0.592	0.652	0.647	0.650	**0.010**	0.139	0.422	0.508	0.443
(100,4)	**520.41**	103.35	58.77	97.57	57.66	**0.315**	0.465	0.569	0.696	0.615	**0.003**	0.101	0.118	0.258	0.134
(100,8)	**497.16**	105.56	41.53	95.15	47.94	**0.367**	0.542	0.625	0.673	0.636	**0.006**	0.170	0.251	0.370	0.279
(100,16)	**464.18**	105	26.54	74.26	44.65	**0.438**	0.602	0.661	0.647	0.655	**0.014**	0.189	0.565	0.636	0.574

Table 12
C-metric of the algorithms ($F = 5$, I denotes INSGAII, K denotes KCA, C denotes CMA, M denotes MOEA/D, N denotes NSGAII).

(*n*,*m*)	*C*(I,K)	*C*(K,I)	*p*	*C*(I,C)	*C*(C,I)	*p*	*C*(I,M)	*C*(M,I)	*p*	*C*(I,N)	*C*(N,I)	*p*
(20,4)	0.985	0.002	0	0.778	0.038	0	1.000	0.000	0	0.663	0.087	0
(20,8)	0.988	0.002	0	0.830	0.027	0	1.000	0.000	0	0.410	0.125	0
(20,16)	0.986	0.001	0	0.937	0.012	0	1.000	0.000	0	0.300	0.125	0
(40,4)	0.945	0.009	0	0.777	0.010	0	1.000	0.000	0	0.990	0.001	0
(40,8)	0.911	0.017	0	0.936	0.004	0	1.000	0.000	0	0.965	0.004	0
(40,16)	0.928	0.011	0	0.999	0.000	0	1.000	0.000	0	0.930	0.007	0
(60,4)	0.906	0.020	0	0.876	0.002	0	1.000	0.000	0	1.000	0.000	0
(60,8)	0.888	0.041	0	1.000	0.000	0	1.000	0.000	0	1.000	0.000	0
(60,16)	0.866	0.029	0	1.000	0.000	0	1.000	0.000	0	1.000	0.000	0
(80,4)	0.903	0.030	0	1.000	0.000	0	1.000	0.000	0	1.000	0.000	0
(80,8)	0.869	0.050	0	1.000	0.000	0	1.000	0.000	0	1.000	0.000	0
(80,16)	0.828	0.044	0	1.000	0.000	0	1.000	0.000	0	1.000	0.000	0
(100,4)	0.893	0.042	0	1.000	0.000	0	1.000	0.000	0	1.000	0.000	0
(100,8)	0.840	0.066	0	1.000	0.000	0	1.000	0.000	0	1.000	0.000	0
(100,16)	0.779	0.076	0	1.000	0.000	0	1.000	0.000	0	1.000	0.000	0

that almost all the non-dominated solutions obtained by the KCA are dominated by those obtained by the INSGAII. The results of the comparison between the INSGAII and CMA (MOEAD, NSGAII) are similar. Comparing the INSGAII with the CMA (MOEA/D, NSGAII), it can be seen that *C*(INSGAII, CMA) is larger than *C*(CMA, INSGAII) on all (n, m) pairs, which means that the non-dominated solutions obtained by INSGAII are better. Since most p values are equal to 0, it means that the INSGAII is significantly better than the KCA, CMA, MOEA/D and NSGAII. Fig. 16 also demonstrates the above conclusion about *C* metric comparisons. Secondly, it can be seen from the results that the *N*-metric of the INSGAII is larger than those of the KCA, CMA, MOEA/D and NSGAII on each pair of (n, m), which means that the INSGAII can obtain more non-dominated solutions than the KCA, CMA, MOEA/D and NSGAII. Furthermore, it can be seen that the *S*-metric and *D*-metric of the INSGAII are the smallest on each pair of (n, m), which means that the diversity and convergence qualities of the solutions obtained by the INSGAII are the best.

Furthermore, we use different CPU time (i.e., $0.25\times n$, $0.5\times n$, and $0.75\times n$ s) to run the algorithms. Fig. 17 shows the Pareto fronts obtained by five algorithms when solving one instance with $F = 2$, $n = 20$, and $m = 8$. Obviously, the Pareto fronts obtained by the INSGAII are always much better in all three cases. Therefore, it can be concluded that INSGAII is more effective than existing algorithms in solving the distributed flowshop scheduling with the criteria to minimize the total flowtime and the total energy consumption.

Table 13
N-metric, *S*-metric, *D*-metric values of the algorithms ($F = 5$, I denotes INSGAII, K denotes KCA, C denotes CMA, M denotes MOEA/D, N denotes NSGAII).

(*n*,*m*)	*N*-metric					*S*-metric					*D*-metric				
	I	K	C	M	N	I	K	C	M	N	I	K	C	M	N
(20,4)	**232.76**	63.52	64.44	61.71	65.42	**0.253**	0.544	0.549	0.603	0.556	**0.000**	0.010	0.014	0.036	0.012
(20,8)	**277.25**	70.25	77.87	70.34	66.32	**0.218**	0.608	0.550	0.617	0.594	**0.000**	0.018	0.036	0.057	0.035
(20,16)	**320.66**	72.67	76.01	74.03	67.15	**0.228**	0.597	0.577	0.587	0.618	**0.001**	0.029	0.087	0.094	0.078
(40,4)	**328.9**	71.63	67.84	83.26	64.62	**0.285**	0.503	0.558	0.689	0.589	**0.001**	0.029	0.036	0.075	0.037
(40,8)	**361.75**	75.3	75.68	94.69	61.3	**0.262**	0.600	0.571	0.660	0.618	**0.001**	0.044	0.079	0.134	0.088
(40,16)	**394.2**	74.49	59.28	97.16	62.41	**0.283**	0.614	0.617	0.618	0.632	**0.003**	0.060	0.185	0.225	0.188
(60,4)	**403.7**	81.34	73.13	97.04	60.81	**0.348**	0.493	0.561	0.697	0.605	**0.002**	0.048	0.057	0.123	0.066
(60,8)	**408.46**	86.98	64.42	104.01	57.25	**0.335**	0.563	0.596	0.674	0.628	**0.003**	0.080	0.119	0.207	0.146
(60,16)	**449.76**	84.47	43.05	101.41	58.49	**0.353**	0.614	0.644	0.637	0.643	**0.005**	0.096	0.294	0.352	0.298
(80,4)	**483.42**	94.11	71.06	106.18	62.76	**0.324**	0.478	0.553	0.699	0.609	**0.002**	0.070	0.080	0.173	0.094
(80,8)	**479.51**	99.56	54.65	109.08	55.99	**0.351**	0.548	0.616	0.676	0.634	**0.004**	0.120	0.173	0.283	0.207
(80,16)	**485.66**	98.65	35.53	101.49	54.82	**0.360**	0.599	0.654	0.644	0.650	**0.008**	0.128	0.409	0.475	0.412
(100,4)	**534.9**	105.18	62.26	113.35	61.59	**0.280**	0.476	0.567	0.695	0.615	**0.003**	0.091	0.106	0.216	0.122
(100,8)	**532.14**	111.53	43.77	111.25	53.58	**0.296**	0.540	0.633	0.674	0.638	**0.007**	0.167	0.243	0.359	0.275
(100,16)	**508.84**	109.16	29.68	94.48	51.25	**0.405**	0.608	0.657	0.644	0.652	**0.013**	0.185	0.552	0.616	0.554

6. Conclusions

In this paper, an improved NSGAII is proposed to solve the EEDPFSP with the minimization of both the total flowtime and the energy consumption criteria. A large number of numerical experiments are used to test the performance of the algorithm. The experimental results show that the new algorithm has better performance than existing algorithms in terms of solution quality and diversity. In addition, the proposed algorithm can obtain high-quality feasible solutions under different stopping criteria.

Future research work can be to try other optimization objectives, or more objectives. In addition, it is also possible to explore solutions based on decomposition, which is another very good type of algorithm in the field of multi-objective optimization. Finally, the application of algorithms to real industrial problems is also a very important research direction. Therefore, our future research work also includes adding some restrictions that meet the actual industrial conditions.

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

Thanks to anonymous reviewers for their comments to improve the quality of this article. This research is partially supported by the National Key Research and Development Program (No.2020YFB1708200), the National Science Foundation of China 61973203, and Shanghai Key Laboratory of Power station Automation Technology.

References

[1] B. Zhang, Q. Pan, L. Gao, X. Li, L. Meng, K. Peng, A multiobjective evolutionary algorithm based on decomposition for hybrid flowshop green scheduling problem, Comput. Ind. Eng. 136 (2019) 325–344.
[2] K. Gao, Y. Huang, A. Sadollah, L. Wang, A review of energy-efficient scheduling in intelligent production systems, Complex Intell. Syst. 6 (2020) 237–249.
[3] S. Irani, G. Singh, S.K. Shukla, R.K. Gupta, An overview of the competitive and adversarial approaches to designing dynamic power management strategies, IEEE Trans. Very Large Scale Integr. (VLSI) Syst. 13 (2005) 1349–1361.
[4] J. Wang, L. Wang, A knowledge-based cooperative algorithm for energy-efficient scheduling of distributed flow-shop, IEEE Trans. Syst. Man Cybern.: Syst. (2019) 1–15.
[5] J. Deng, L. Wang, C. Wu, J. Wang, X. Zheng, A competitive memetic algorithm for carbon-efficient scheduling of distributed flow-shop, in: D.S. Huang, V. Bevilacqua, P. Premaratne (Eds.), Lecture Notes in Computer Science, 2016.
[6] Q. Zhang, H. Li, MOEA/D: A multiobjective evolutionary algorithm based on decomposition, IEEE Trans. Evol. Comput. 11 (2007) 712–731.
[7] K. Deb, A. Pratap, S. Agarwal, T. Meyarivan, A fast and elitist multiobjective genetic algorithm: NSGA-II, IEEE T. Evolut. Comput. 6 (2002) 182–197.
[8] B. Naderi, R. Ruiz, The distributed permutation flowshop scheduling problem, Comput. Oper. Res. 37 (2010) 754–768.
[9] 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.
[10] Y. Xu, L. Wang, S. Wang, M. Liu, An effective hybrid immune algorithm for solving the distributed permutation flow-shop scheduling problem, Eng. Optim. 46 (2014) 1269–1283.
[11] 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.
[12] S. Lin, K. Ying, C. Huang, Minimising makespan in distributed permutation flowshops using a modified iterated greedy algorithm, Int. J. Prod. Res. 51 (2013) 5029–5038.
[13] J. Gao, R. Chen, Y. Liu, A knowledge-based genetic algorithm for permutation flowshop scheduling problems with multiple factories, Int. J. Adv. Comput. Technol. 4 (2012) 121–129.
[14] S. 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 (2013) 387–396.
[15] B. Naderi, R. Ruiz, A scatter search algorithm for the distributed permutation flowshop scheduling problem, European J. Oper. Res. 239 (2014) 323–334.
[16] H. Bargaoui, O. Belkahla 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.
[17] R. Ruiz, Q. Pan, B. Naderi, Iterated greedy methods for the distributed permutation flowshop scheduling problem, Omega 83 (2019) 213–222.
[18] 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.
[19] J. Huang, Q. 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) 100742.
[20] X. Jing, Q. Pan, L. Gao, Y. Wang, An effective iterated greedy algorithm for the distributed permutation flowshop scheduling with due windows, Appl. Soft Comput. 96 (2020) 106629.
[21] Q. Pan, L. Gao, L. Wang, J. Liang, X. Li, Effective heuristics and metaheuristics to minimize total flowtime for the distributed permutation flowshop problem, Expert Syst. Appl. 124 (2019) 309–324.
[22] J. Huang, Q. Pan, Z. Miao, L. Gao, Effective constructive heuristics and discrete bee colony optimization for distributed flowshop with setup times, Eng. Appl. Artif. Intel. 97 (2021) 104016.
[23] W. Shao, D. Pi, Z. Shao, Optimization of makespan for the distributed no-wait flow shop scheduling problem with iterated greedy algorithms, Knowl.-Based Syst. 137 (2017) 163–181.
[24] S. Lin, K. Ying, Minimizing makespan for solving the distributed no-wait flowshop scheduling problem, Comput. Ind. Eng. 99 (2016) 202–209.
[25] Y. Li, Q. Pan, J. Li, L. Gao, M.F. Tasgetiren, An adaptive iterated greedy algorithm for distributed mixed no-idle permutation flowshop scheduling problems, Swarm Evol. Comput. 63 (2021) 100874.

[26] K. Ying, S. Lin, C. Cheng, C. He, Iterated reference greedy algorithm for solving distributed no-idle permutation flowshop scheduling problems, Comput. Ind. Eng. 110 (2017) 413–423.
[27] J. Li, J. Deng, C. Li, Y. Han, J. Tian, B. Zhang, C. Wang, An improved Jaya algorithm for solving the flexible job shop scheduling problem with transportation and setup times, Knowl.-Based Syst. (2020).
[28] Y. Han, J. Li, D. Gong, H. Sang, Multi-objective migrating birds optimization algorithm for stochastic lot-streaming flow shop scheduling with blocking, IEEE Access 7 (2019) 5946–5962.
[29] K. Ying, S. Lin, Minimizing makespan in distributed blocking flowshops using hybrid iterated greedy algorithms, IEEE Access 5 (2017) 15694–15705.
[30] J. Li, P. Duan, H. Sang, S. Wang, Z. Liu, P. Duan, An efficient optimization algorithm for resource-constrained steelmaking scheduling problems, IEEE Access 6 (2018) 33883–33894.
[31] Q. Pan, L. Gao, L. Wang, An effective cooperative co-evolutionary algorithm for distributed flowshop group scheduling problems, IEEE T. Cybernetics (2020).
[32] S. Hatami, R. Ruiz, C. Andres-Romano, The distributed assembly permutation flowshop scheduling problem, Int. J. Prod. Res. 51 (2013) 5292–5308.
[33] J. Li, X. Tao, B. Jia, Y. Han, C. Liu, P. Duan, Z. Zheng, H. Sang, Efficient multi-objective algorithm for the lot-streaming hybrid flowshop with variable sub-lots, Swarm Evol. Comput. 52 (2020) 100600.
[34] J. Ding, S. Song, C. Wu, Carbon-efficient scheduling of flow shops by multi-objective optimization, European J. Oper. Res. 248 (2016) 758–771.
[35] X. Li, S. Ma, Multiobjective discrete artificial bee colony algorithm for multiobjective permutation flow shop scheduling problem with sequence dependent setup times, IEEE Trans. Eng. Manage. 64 (2017) 149–165.
[36] Y. Fu, G. Tian, A.M. Fathollahi-Fard, A. Ahmadi, C. Zhang, Stochastic multi-objective modelling and optimization of an energy-conscious distributed permutation flow shop scheduling problem with the total tardiness constraint, J. Cleaner Prod. 226 (2019) 515–525.
[37] Z. Shao, D. Pi, W. Shao, A multi-objective discrete invasive weed optimization for multi-objective blocking flow-shop scheduling problem, Expert Syst. Appl. 113 (2018) 77–99.
[38] D. Yüksel, M.F. Taşgetiren, L. Kandiller, L. Gao, An energy-efficient bi-objective no-wait permutation flowshop scheduling problem to minimize total tardiness and total energy consumption, Comput. Ind. Eng. 145 (2020) 106431.
[39] H. Öztop, M.F. Tasgetiren, D.T. Eliiyi, Q. Pan, L. Kandiller, An energy-efficient permutation flowshop scheduling problem, Expert Syst. Appl. 150 (2020) 113279.
[40] E. Jiang, L. Wang, An improved multi-objective evolutionary algorithm based on decomposition for energy-efficient permutation flow shop scheduling problem with sequence-dependent setup time, Int. J. Prod. Res. 57 (2018) 1756–1771.
[41] J. Li, H. Sang, Y. Han, C. Wang, K. Gao, Efficient multi-objective optimization algorithm for hybrid flow shop scheduling problems with setup energy consumptions, J. Cleaner Prod. 181 (2018) 584–598.
[42] B. Zhang, Q. Pan, L. Gao, L. Meng, X. Li, K. Peng, A three-stage multiobjective approach based on decomposition for an energy-efficient hybrid flow shop scheduling problem, IEEE Trans. Syst. Man Cybern.: Syst. 50 (2020) 4984–4999.
[43] J. Deng, L. Wang, A competitive memetic algorithm for multi-objective distributed permutation flow shop scheduling problem, Swarm Evol. Comput. 32 (2017) 121–131.
[44] S.F. Rad, R. Ruiz, N. Boroojerdian, New high performing heuristics for minimizing makespan in permutation flowshops, Omega 37 (2009) 331–345.