Engineering Applications of Artificial Intelligence 78 (2019) 124–141

Contents lists available at ScienceDirect

Engineering Applications of Artificial Intelligence

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

An efficient discrete invasive weed optimization for blocking flow-shop scheduling problem

Zhongshi Shao [a], Dechang Pi [a,b,*], Weishi Shao [a], Peisen Yuan [c]

[a] *College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing, China*
[b] *Collaborative Innovation Center of Novel Software Technology and Industrialization, Nanjing, China*
[c] *College of Information Science and Technology, Nanjing Agricultural University, Nanjing, China*

A R T I C L E I N F O

Keywords:
Flow-shop scheduling with blocking
Makespan
Invasive weed optimization
Spatial dispersal
Local search

A B S T R A C T

This paper proposes a discrete invasive weed optimization (DIWO) to solve the blocking flow-shop scheduling problem (BFSP) with makespan criterion, which has important practical applications in modern industry. In the proposed DIWO, an effective heuristic and the random method are combined to generate an initial plant population with high quality and diversity. To keep the searching ability and efficiency, a random-insertion-based spatial dispersal is presented by means of the normal distribution. Moreover, a shuffle-based referenced local search is embedded to further enhance local exploitation ability. An improved competitive exclusion is developed to determine an offspring plant population with good quality and diversity. The parameters setting is investigated based on a design-of-experiment approach. The effectiveness and applicability of the proposed spatial dispersal and local search are confirmed through numerical comparisons. Finally, a comprehensive computational evaluation including several state-of-the-art algorithms, together with statistical analyses, show that the proposed DIWO algorithm produces better results than all compared algorithms by significant margin.

1. Introduction

The blocking flow-shop scheduling problem (BFSP) (Pinedo, 2012) is an important branch of the traditional flow-shop scheduling problems. In such problem, due to technological requirements or process characteristics in some stages of manufacturing process, there are no intermediate buffers or buffers are not allowed between the adjacent machines so that the processed jobs must be kept in the current machines until the downstream machines are free (Wang et al., 2010b). Many industrial productive systems can be modeled as BFSP, e.g., chemical industry (Merchan and Maravelias, 2016; Ronconi, 2004), serial manufacturing processes (Koren et al., 2017), the production of concrete blocks (Grabowski and Pempera, 2000), robotic cell (Ribas et al., 2015), the iron and steel industry (Gong et al., 2010), etc. In view of the practical significance of the BFSP, it is essential to develop effective methods to solve it.

The BFSP is a typical $\mathcal{NP}$-hard problem (Hall and Sriskandarajah, 1996) when the number of machines is larger than two. With the increase of the problem size, the BFSP becomes more and more complicated so that it is difficult to completely solve it. To tackle this challenge, much more effort has been dedicated to providing effective methods to find high-quality solutions instead of the optimal solutions in a reasonable computational time. These effective methods can be classified into two categories: constructive heuristics and improvement heuristics. For the constructive heuristics, some specific rules are used to assign a priority index to each job to construct a scheduling permutation. Some constructive heuristics have been proposed to solve the BFSP, e.g., Profile Fitting (PF) (Mccormick et al., 1989), MinMax (MM) (Ronconi, 2004), etc. To further get the better solutions, Ronconi (2004) incorporated the insertion procedure of NEH into the MM and PF heuristics to propose two improved heuristics, i.e., MME and PFE. Meanwhile, Ronconi showed that the performance of both MME and PFE was better than the original NEH over the problems with 500 jobs and 20 machines. Pan and Wang (2012) found that the idle time and blocking time on the earlier stage machines may lead to more delay for the successive jobs. According to this rule, two constructive heuristics with weighted values, i.e., wPF and PW were proposed. Meanwhile, in order to retain the good characteristics of the PF and PW heuristics, they also combined them with the partial NEH implementation to develop three new constructive heuristics including PF-NEH, wPF-NEH and PW-NEH. Besides those above, Wang et al. (2012) considered the average value and standard deviation of processing time to propose a modified NEH heuristic. Ribas et al. (2011) presented a modified NEH heuristic which considered the reversibility of the problems. Fernandez-Viagas et al. (2016) conducted a comprehensive evaluation of the available constructive heuristics for BFSP, and proposed a constructive heuristic based on beam search approach.

* Corresponding author.
E-mail addresses: shaozhongshi@hotmail.com (Z. Shao), dc.pi@nuaa.edu.cn (D. Pi), shaoweishi@hotmail.com (W. Shao), peiseny@njau.edu.cn (P. Yuan).

https://doi.org/10.1016/j.engappai.2018.11.005
Received 11 February 2017; Received in revised form 5 August 2018; Accepted 15 November 2018
Available online 27 November 2018

For the improvement heuristics, many novel metaheuristics recently have been proposed and provided excellent results in acceptable computational time. Wang et al. (2010a) proposed a hybrid discrete differential evolution (HDDE) algorithm. In the HDDE, the new mutation and crossover operators were developed. A local search based on the insert neighborhood was embedded to enhance the local exploitation ability. A speedup method was presented to evaluate the insert neighboring solutions. This speedup method effectively improved the efficiency of the whole algorithm. Wang and Tang (2012) proposed a discrete particle swarm optimization (DPSO) in which a self diversity control strategy was adopted to diversify the population, and a stochastic variable neighborhood search was used to improve the search intensification. Lin and Ying (2013) proposed a revised artificial immune system (RAIS) based on the characteristics of artificial immune systems and the annealing process of simulated annealing algorithm. Pan et al. (2013b) proposed a high-performing memetic algorithm (MA) which was consisted of an effective heuristic combined PF with NEH, a path-relinking-based crossover operator, a referenced local search and a procedure of controlling diversity. Ribas et al. (2011, 2013) proposed two effective non-population-based algorithms: iterated greedy algorithm (IG) and competitive variable neighborhood search (SVNS) algorithm. Ding et al. (2016) investigated some new blocking properties of the BFSP and proposed an iterated greedy (IG) algorithm based on these priorities. Han et al. (2016a) proposed a modified fruit fly optimization (MFFO) algorithm in which a problem-specific heuristic, a smell-based search, and a speed-up insert-neighborhood-based local search were employed. Apart from the metaheuristics above, some methods were also presented to solve the BFSP with makespan criterion, e.g., an estimation of distribution algorithm (Jarboui et al., 2009), an improved artificial bee colony (IABC) (Han et al., 2011), a hybrid modified global-best harmony search (hmgHS) (Wang et al., 2011), a dynamic multi-swarm particle swarm optimizer (DMS-PSO) (Liang et al., 2011), a cluster enhanced differential evolution (EDEc) (Davendra et al., 2012), a three-phase algorithm (TPA) (Wang et al., 2012), a discrete self-organizing migrating algorithm (DSOMA) (Davendra and BialicDavendra, 2013), a discrete artificial bee colony algorithm incorporating differential evolution (DE-ABC) (Han et al., 2015b), a populated local search with differential evolution algorithm (DE_PLS) (Tasgetiren et al., 2015), a hybrid combinatorial particle swarm optimization algorithm (HCPSO) (Eddaly et al., 2016), iterated greedy algorithms (Tasgetiren et al., 2017), an estimation of distribution with path relinking (P-EDA) (Shao et al., 2018a), etc. Besides, many other effective metaheuristics have been also presented to solve other types of BFSP in recent years. Riahi et al. (2017) employed the scatter search to solve the mixed BFSP. Shao et al. (2018b) proposed an discrete water wave optimization (DWWO) algorithm to solve the BFSP with sequence-dependent setup times. Nouri and Ladhari (2017) proposed a genetic algorithm (MBGA) for BFSP with makespan and total flowtime. For the same problem, Deng et al. (2016) proposed a multi-objective discrete group search optimizer (MDGSO). Shao et al. (2017) presented a self-adaptive discrete invasive weed optimization (SaDIWO) to solve the BFSP with total tardiness criterion. Aiming at the same problem, Nagano et al. (2017) also presented an evolutionary clustering search (ECS) algorithm. Although many methods have been proposed and applied to solve variety of BFSPs, the pursuit of efficient methods should not be stopped.

Invasive weed optimization (IWO) is a swarm intelligence optimization algorithm, which was proposed by Mehrabian and Lucas (2006) for solving the solution of complex real world problems. IWO mimics the ecological behavior of colonizing weeds and has good exploration and exploitation ability in the search area. It shares many characteristics with evolutionary algorithms, but does not utilize evolution operators such as crossover and mutation. Instead, some interesting operators based on the features of weed plants, including reproduction, spatial dispersal, and competitive exclusion are employed. As a more robust, stochastic and derivative-free optimization tool (Barisal and Prusty, 2015), IWO has been successfully applied to solve many optimization problems, e.g., production scheduling (Sang et al., 2018a), non-uniform circular antenna arrays (Roy et al., 2011), economic load dispatch (Barisal and Prusty, 2015), traveling salesman problem (Zhou et al., 2015), unit commitment problem solution (Saravanan et al., 2014), portfolio optimization problem (Rezaei Pouya et al., 2016), distributed generation (Rama Prabha and Jayabarathi, 2016), antenna design (Dastranj, 2017), vendor selection (Niknamfar and Niaki, 2018), optimal chiller loading (Zheng and Li, 2018), etc. Therefore, in view of the merits and extensive applications of IWO, we propose an effective discrete invasive weed optimization (DIWO) to solve the BFSP with makespan criterion in this paper. In the proposed algorithm, an effective heuristic and the random method are combined to generate an initial plant population. A random-insertion-based spatial dispersal is designed to produce the new individuals with high quality. A shuffle-based referenced local search is incorporated to reinforce the exploitation ability. Finally, to obtain an offspring plant population with high quality and diversity, an improved competitive exclusion is employed. Additionally, to the best of our knowledge, this work is the first reported application of IWO to solve the BFSP with makespan criterion. The major contribution and innovation of our work can be reflected in algorithm design and experimental results.

- A discrete invasive weed optimization (DIWO) is proposed for solving BFSP with makespan criterion in this study. The phases of reproduction, spatial dispersal, and competitive exclusion are newly customized. Especially, a random-insertion-based spatial dispersal is designed to make the original IWO for solving the continuous optimization problems apply to the discrete optimization problem. More importantly, we incorporate a shuffle-based referenced local search into DIWO, which effectively enhances the performance of the algorithm. This new phase can be regarded as a supplement for the framework of the original IWO, which also provides an idea for further improving the performance of IWO.
- IWO is a nature-inspired optimization algorithm in artificial intelligence, which is initially used to solve the continuous optimization problems on engineering area. In this study, the IWO is successfully extended to solve a typical combinatorial optimization problem, which causes an enhancement in understanding of the searching behavior of IWO, and further enriches its applications. Meanwhile, the proposed discrete IWO incorporating distinct reproduction, spatial dispersal and competitive exclusion mechanism can also provide a solution for solving the similar engineering problems.
- The performance of the proposed DIWO algorithm was evaluated and compared with several high-performing methods in recent literature. The experimental results demonstrated that our proposed algorithm was superior to these methods. Additionally, we have summarized the best solutions of 14 recent state-of-the-art algorithms and compared with the proposed DIWO. The comparison results show that the proposed DIWO updates 47 best solutions out of 120 Taillard's benchmark instances (Taillard, 1993). The job processing sequences of these new best solutions are provided on the online supplementary material, which can be a reference for future work along this line of research.

The remainder of this paper is organized as follows. In Section 2, the BFSP problem is described. Next, the basic IWO is presented in Section 3. Section 4 elaborates on the proposed DIWO algorithm. Section 5 gives a full performance evaluation and comparison. Finally, Section 6 provides the concluding remarks and future research directions.

2. Blocking flow-shop scheduling problem

The blocking flow-shop scheduling problem (BFSP), denoted as $Fm|blocking|C_{\max}$ according to Graham et al. (1979) can be briefly described as: n jobs have to be processed m machines with the same permutation on each machine. There are no intermediate buffers between any two consecutive machines. Due to no buffers between the adjacent machines, a job having its operation on the current machine cannot leave the machine until the next machine is free for processing. That is to say, if the downstream machine is busy, the job must be blocked in the current machine. Additionally, other constraints should be considered for Fm|blocking|$C_{\max}$, which are described as follows:

- Each job can be only processed by one machine at a time.
- Each machine can process only one job at a time.
- The processing time of each job on each machine is predetermined.
- The setup time and transportation time are assumed to be included in the processing time.
- Preemption is not allowed.

The purpose considered in Fm|blocking|$C_{\max}$ is to obtain a permutation for processing all jobs on all machines so that its maximum completion time (makespan, denoted as $C_{\max}$) is minimized. Let $\pi = [\pi(1), \pi(2), \ldots, \pi(n)]$ be a permutation sequence. $p_{\pi(i),j}$ is the processing time of job $\pi(i)$ on machine j. $d_{\pi(i),j}$ denotes the departure time of job $\pi(i)$ on machine j. According to Ronconi (2004), $d_{\pi(i),j}$ can be calculated as follows:

$$d_{\pi(1),0} = 0 \tag{1}$$

$$d_{\pi(1),j} = d_{\pi(1),j-1} + p_{\pi(1),j} \quad j = 1, \ldots, m-1 \tag{2}$$

$$d_{\pi(i),0} = d_{\pi(i-1),1} \quad i = 2, \ldots, n \tag{3}$$

$$d_{\pi(i),j} = max\left\{d_{\pi(i),j-1} + p_{\pi(i),j}, d_{\pi(i-1),j+1}\right\} \quad i = 2, \ldots, n \quad j = 1, \ldots, m-1 \tag{4}$$

$$d_{\pi(i),m} = d_{\pi(i),m-1} + p_{\pi(i),m} \quad i = 1, \ldots, n \tag{5}$$

where $d_{\pi(1),0}$ denotes the beginning time of processing. Then, the makespan or the maximum completion time of the schedule $\pi = [\pi(1), \pi(2), \ldots, \pi(n)]$ is $C_{\max}(\pi) = \max_{i=1,2,\ldots,n} d_{\pi(i),m} = d_{\pi(n),m}$. Therefore, the BFSP with makespan criterion is to find a job permutation $\pi^* \in \mathbf{\Pi}$ such that $C_{\max}(\pi^*) = \min_{\pi \in \mathbf{\Pi}} C_{\max}(\pi)$, where $\mathbf{\Pi}$ is the set of all job permutations.

3. Introduction to the basic IWO algorithm

The Invasive Weed Optimization (IWO) is a swarm intelligence metaheuristic, which mimics the natural behavior of weeds in colonizing and finding suitable place for growth and reproduction (Basak et al., 2013). Some interesting properties of weeds that are invasive, fast reproduction, distribution and competitive exclusion are taken advantages by the IWO. The basic IWO is composed of four basic steps: initialization, reproduction, spatial dispersal and competitive exclusion. They are described as follows (Mehrabian and Lucas, 2006):

(1) **Initialization**. A population $\boldsymbol{POP} = \{X_1, X_2, \ldots, X_{N_0}\}$, $X_i = [x_{i,1}, x_{i,2}, \ldots, x_{i,D}]$ that has N_0 individuals (weeds) is randomly generated over the D-dimensional search space.

(2) **Reproduction**. Each weed of plant population can generate some seeds within a specified region centered at its own position. The number of seeds for each weed is determined by its relative fitness with respect to the best and the worst fitness, which can be calculated by:

$$S_i = \left\lfloor \frac{f_{\max} - f_i}{f_{\max} - f_{\min}} \times \left(S_{\max} - S_{\min}\right) + S_{\min} \right\rfloor \tag{6}$$

where S_i is the number of seeds for ith weed. f_i is the fitness value of ith weed. $f_{\min}$ and $f_{\max}$ denote the fitness values of the best and worst weeds among the current population, respectively. $S_{\min}$ and $S_{\max}$ denote the minimum and maximum number of seeds that allowed to be generated for each weed, respectively. It is clear from Eq. (6) that the higher the fitness f_i is, the more seeds it produces.

(3) **Spatial dispersal**. The generated seeds are randomly scattered over the D-dimensional search space by means of a normal distribution with mean zero and varying standard variance σ. The normal distribution ensures that the generated seeds are scattered near to their parents. σ adaptively decreases from $\sigma_{\max}$ to $\sigma_{\min}$ with the increase of generation so that the algorithm can gradually move from exploration to exploitation. The standard variance for the kth iteration σ_k is given by:

$$\sigma_k = \left(\frac{iter_{\max} - k}{iter_{\max}}\right)^n \times (\sigma_{\max} - \sigma_{\min}) + \sigma_{\min} \tag{7}$$

where n is a nonlinear modulation index. $iter_{\max}$ is the maximum number of iterations. k is the current iteration number. In addition, according to the standard variance σ_k above, the location of a seed are determined by:

$$x'_{i,j} = x_{i,j} + N(0, \sigma_k^2) \tag{8}$$

where $x_{i,j}$ is the jth dimension of the ith weed in current population. $N(0, \sigma_k^2)$ is a sampling function that returns a normally distributed random number with mean zero and the standard deviation σ_k.

(4) **Competitive exclusion**. After all seeds have found their locations over the search area, the new seeds would grow to weeds. Initially, the weeds in a colony will reproduce fast and all produced weeds will be reserved in the colony. When the number of weeds in a colony reaches its maximum value ($P_{\max}$), all new created weeds and their parental weeds are ranked together according to their fitness. The weaker and inappropriate weeds are eliminated so that the colony can maintain the maximum allowable population. This mechanism gives a chance for lower fitness plants to reproduce. If their offspring individuals have good fitness, then they can survive. This procedure is named competitive exclusion and is the selection procedure of IWO.

4. DIWO for BFSP

From the procedure of the basic IWO above, it can be seen that IWO can be directly applied only when solving continuous optimization problems. That is to say, when solving the discrete optimization problems, a discrete variant of IWO must be developed. Therefore, a discrete IWO (DIWO) is presented to solve BFSP with makespan criterion in this section. The proposed DIWO algorithm contains four main components: a population scheme combining an effective heuristic and the random method to produce an initial plant population with high quality and diversity, a spatial dispersal model based on random insertion to maintain the searching ability, a competitive exclusion mechanism to determine the offspring individuals and an effective local search procedure to enhance the performance of DIWO. With these effective and advanced technologies, the DIWO is expected to generate high quality solutions with robustness for BFSP. In the following sections, each component of DIWO is firstly elaborated, and then we outline the complete procedure of DIWO.

4.1. Solution representation and initialization

The permutation-based-jobs representation is used to encode individuals, which has been frequently adopted in the literature (Han et al., 2016a; Ribas et al., 2015; Shao and Pi, 2016). Given a permutation of jobs, the jobs are successively scheduled from left to right onto machines. To generate an initial plant population with a certain level of quality and diversity, a well-known heuristic named PF-NEH(x) (Pan and Wang, 2012) is firstly employed to produce a weed with high quality. Then, the rest weeds are randomly generated in the entire colony. PF-NEH(x) not only considers the total idle and blocking time but also the different initial jobs. The procedure of initialization is described in Algorithm 1.

Following Pan and Wang (2012), the PF-NEH(x) with $\lambda = 25$ and $x = 5$ can obtain the best performance. Moreover, it should be noted that the speedup method proposed by Wang et al. (2010a) is used to evaluate the $((2n - \lambda + 1)\lambda)/2$ partial sequences in the insertion phase of the NEH heuristic, so the computational complexity of NEH can be reduced from $O(mn^3)$ to $O(mn^2)$. Since x sequences are generated in PF-NEH(x), the computational complexity is $O(xmn^2)$.

Algorithm 1. Population initialization

```
Input: $N_0$, $\lambda$ and $x$
Output: $\boldsymbol{POP}$
01: $\boldsymbol{POP} \leftarrow \emptyset$;
02: $\pi_I \leftarrow$ Generate an initial order of all jobs according to their non-decreasing total processing time;
03: $k \leftarrow 1$, $\boldsymbol{\Gamma} \leftarrow \emptyset$;
04: repeat
05:     Set $\pi'(1) = \pi_I(k)$ and the unscheduled job set $\boldsymbol{U} = \boldsymbol{J} \backslash \{\pi_I(k)\}$, where $\boldsymbol{J}$ is the job set;
06:     Calculate the departure time $d_{1,j}$ for the position 1 of $\pi'$, where $j = 1, \ldots, m$;
07:     $i \leftarrow 2$;
08:     while $\boldsymbol{U} \neq \emptyset$ do
09:         Compute the departure time $d_{i,j}$ of each job in U for position $i$, $j = 1, \ldots, m$;
10:         Compute the sum of the idle time and blocking time of each job in U for position $i$ by:
                $v_l = \sum_{j=1}^{m} \left( d_{i,j} - d_{i-1,j} - p_{\pi_l, j} \right) \quad l \in \boldsymbol{U}$        (9)
11:         $\pi'(i) \leftarrow$Select the job with the smallest of $v_l$ in U;
12:         $i \leftarrow i + 1$ and $\boldsymbol{U} \leftarrow \boldsymbol{U} \backslash \{\pi_{\min v_l}\}$;
13:     end while
14:     $\pi'' \leftarrow [\pi'(1), \pi'(2), \ldots, \pi'(n - \lambda)]$;
15:     for $q = n - \lambda + 1$ to $n$ do
16:         Take the job $\pi'(q)$ from $\pi'$ and test it into all possible positions of $\pi''$;
17:         Insert the job $\pi'(q)$ into the position of $\pi''$ with the lowest makespan;
18:     end for
19:     $k \leftarrow k + 1$ and $\boldsymbol{\Gamma} \leftarrow \boldsymbol{\Gamma} \cup \{\pi''\}$;
20: until $k == x$
21: $\pi_{best} \leftarrow$Select the best one from $\boldsymbol{\Gamma}$;
22: $\boldsymbol{POP} \leftarrow \boldsymbol{POP} \cup \{\pi_{best}\}$
23: repeat
24:     $\pi \leftarrow$Randomly produce a weed in the entire colony;
25:     If $\boldsymbol{POP} \cap \{\pi\} == \emptyset$, then $\boldsymbol{POP} \leftarrow \boldsymbol{POP} \cup \{\pi\}$;
26: until $|\boldsymbol{POP}| == N_0$.
```

4.2. *Reproduction*

In the procedure of the basic IWO, the number of seeds generated by each weed is determined by its fitness. The higher the fitness is, the larger the number is. However, since the BFSP is a discrete optimization problem and the solution space is limited, many different permutations have the same fitness. When the fitness of each individual in population is same but they have different permutation sequences, the best fitness is equal to the worst fitness so that Eq. (6) is illegal. Actually, this illegal case often appears when solving the small instances, it is because the solution space is relatively small so that the individuals in the population quickly converge to a local or global optimum. Therefore, to circumvent such minor problem, the value of S_i produced by the weed π_i is determined by the following formula:

$$S_i = \left\lfloor \frac{C_{\max}\left(\pi_{worst}\right) - C_{\max}\left(\pi_i\right) + \epsilon}{C_{\max}(\pi_{worst}) - C_{\max}(\pi_{best}) + \epsilon} \times \left(S_{\max} - S_{\min}\right) + S_{\min} \right\rfloor \tag{10}$$

where ϵ is the smallest constant in the computer, which is used to avoid division-by-zero. π_{worst} and π_{best} denote the worst and best weeds in the current plant population, respectively. $S_{\min}$ and $S_{\max}$ denote the minimum and maximum number of seeds, respectively. $f\left(\pi_i\right)$ denotes the makespan of π_i.

4.3. *Random-insertion-based spatial dispersal*

In the basic IWO, the dispersion of new seeds to the whole colony primarily depends on the normal distribution with mean zero and varying standard deviation. The varying standard deviation σ_k has a huge effect on the performance of the algorithm. However, such mechanism has its own shortcomings. The first one is, all weeds have same σ_k so that the spread scope of all new generated seed is same. As a result, the weeds with lower fitness may not get more opportunities to evolve to the better ones, while the weeds near the global optimum may be trapped into local optimum. The second one is, the maximum CPU execution time is usually adopted as the stopping rule in the many scheduling literature (Ding et al., 2016; Pan et al., 2013b; Ribas et al., 2015), whereas the standard deviation σ_k of the basic IWO algorithm is adjusted based on the iterations. Moreover, it is difficult to exactly predict the number of iterations in the finite execution time. Therefore, a mechanism based on fitness and execution time is proposed to overcome these shortcomings. Actually, the computational time of each iteration is approximate, so we can estimate an approximate total number of iterations. On the basis of this principle, a decreased σ_k with execution time is firstly given in Eq. (11) to solve the second problem.

$$\sigma_k = \left(1 - \frac{t - t_0}{t_{\max}}\right) \times \left(\sigma_{\max} - \sigma_{\min}\right) + \sigma_{\min} \tag{11}$$

where t is the current time. $t_{\max}$ is the maximum CPU execution time. t_0 is the beginning time of the algorithm. $\sigma_{\min}$ and $\sigma_{\max}$ denote the minimum and maximum values of σ_k, respectively. When any weeds are close to the potential global optimum, σ_k would decrease as Eq. (11) so that it does not miss the position of true global optimum. Meanwhile, the weeds with lower fitness should have larger spread scope to explore other promising areas. Thus, for the ith weed of the plant population in the kth iteration, $\sigma_{i,k}$ is calculated by:

$$\sigma_{i,k} = \begin{cases} \sigma_k & f\left(\pi_i\right) < f\left(\pi_{median}\right) \\ \sigma_k \times \left(\dfrac{C_{\max}\left(\pi_i\right) - C_{\max}\left(\pi_{median}\right)}{C_{\max}\left(\pi_{worst}\right) - C_{\max}\left(\pi_{median}\right) + \epsilon} \times 0.5 + 1\right) & f\left(\pi_i\right) > f\left(\pi_{median}\right) \end{cases} \tag{12}$$

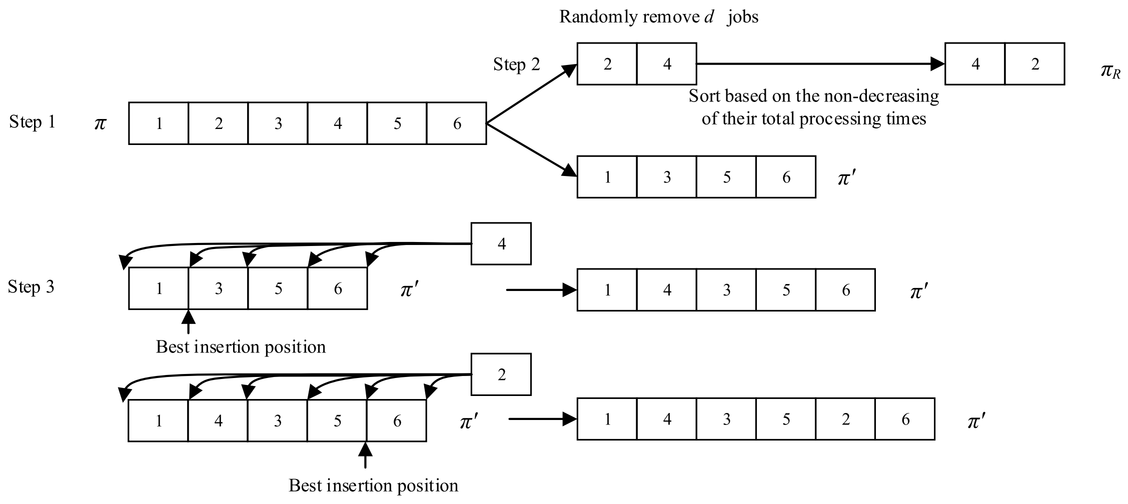

Fig. 1. Numerical example of the random insertion neighborhood.

where π_{median} denotes the median element of the sorted plant population (sorted ascending according to fitness values). π_{worst} is the worst weed in the current plant population. ε is the smallest number in the computer, which is used to avoid division-by-zero. As can be easily observed from Eq. (12), when the makespan of π_i is smaller than $C_{\max}\left(\pi_{median}\right)$, $\sigma_{i,k}$ is set to 1.5 times σ_k, which gives a chance to the worse weeds to spread their weeds over wider range and find more promising individuals.

Since the BFSP is a discrete problem and the solutions are represented by the job permutations, the spatial dispersal of the basic IWO is not suitable. Therefore, we develop a discrete spatial dispersal. To be specific, in our proposed spatial dispersal, a random insertion neighborhood is designed to generate a new seed for each weed, which has three steps. Firstly, d jobs are randomly selected and removed from the solution (weed) π without repetition to form a partial sequence π_R. The remainder jobs in π form another partial sequence π'. Afterwards, according to Wang et al. (2011), the jobs with higher total processing times may cause to block their successive jobs and yield larger blocking times than less total processing times. The blocking time would increase the makespan value. The jobs with less total processing time should have high priorities. So we sort the jobs in π_R with the non-decreasing of their total processing time. Lastly, the jobs in π_R are successively re-inserted into π' by using the insertion of the NEH heuristic until a complete solution π' of n jobs is re-established. For the insertion position of each job, all possible insertion positions of π' are evaluated and the best one is chosen.

A numerical example of the random insertion neighborhood is shown in Fig. 1. The jobs 2 and 4 are firstly removed from $\pi = [1, 2, 3, 4, 5, 6]$ to construct two partial sequences $\pi_R = [2, 4]$ and $\pi' = [1, 3, 5, 6]$. Then, sort the jobs in π_R according to the non-decreasing of their total processing time, as a result, $\pi_R = [4, 2]$. Next, we try to insert the job 4 into each possible position of π'. The position with the lowest makespan (i.e., the position 2) is selected and the job 4 is inserted into it. Similarly, the job 2 is inserted into the position 5 of π'. Hence, a new seed $\pi' = [1, 4, 3, 5, 2, 6]$ is generated.

The number of the removed jobs, i.e., d, determines the performance of the discrete spatial dispersal. A larger d ($d \le n$) can produce a larger neighborhood to explore the wider range, while a smaller d can be used to exploit the areas around the current solutions. Therefore, we draw support from the normal distribution and employ the proposed varying standard deviation $\sigma_{i,k}$ above to adjust d_i for each weed of the plant population. d_i is defined as:

$$d_i = \left\lfloor \left| N(0, \sigma_{i,k}^2) \right| \right\rfloor \tag{13}$$

where $N(0, \sigma_{i,k}^2)$ is a sampling function which returns a normally distributed random number with mean zero and the standard deviation $\sigma_{i,k}$. However, when a random number is sampled from the normal distribution, it may be close to the length of π or larger than it. If d_i is close to n, the proposed spatial dispersal would be simplified as NEH, which degrades its performance. Hence, d_i is restricted by the following way:

$$d_i = \lfloor \sigma_{\min} + rand \times (\sigma_{\max} - \sigma_{\min}) \rfloor \qquad \text{if} \quad d_i > \frac{n}{2} \text{ or } d_i < \sigma_{\min} \tag{14}$$

where $rand$ is a uniformly random number between 0 and 1. Eq. (14) makes d_i restrict in a reasonable area. $\sigma_{\min}$ and $\sigma_{\max}$ would be calibrated in Section 5.2.

The detailed procedure of the proposed spatial dispersal is given in Algorithm 2. Note that lines 01–03 are the procedure of reproduction. The main computational burden in this spatial dispersal lies in lines 14–17, where all possible insertions have to be evaluated. The same speedup method as in the implementation of the PF-NEH(x) heuristic is employed to evaluate these insertions, so the generation of a new seed can be completed in $O(dm)$.

4.4. Local search

To strengthen the local exploitation ability of the proposed DIWO algorithm, a local search is presented and embedded to quickly guide the individuals toward the best area. The referenced local search (RLS) proposed by Wang et al. (2010a) has been employed by many algorithms (Pan et al., 2013b; Tasgetiren et al., 2017; Wang et al., 2011) to solve the BFSP. In the RLS algorithm, $\pi_r = [\pi_r(1), \pi_r(2), \ldots, \pi_r(n)]$ is a reference sequence. The best solution so far is usually selected as the reference sequence. Let $\pi_c = [\pi_c(1), \pi_c(2), \ldots, \pi_c(n)]$ denote the current individual. The procedure of RLS can be briefly described as follows: (1) remove the job $\pi_r(i)$ from the current sequence π_c and test it in all positions of π_c; (2) find the position with the lowest makespan and re-insert $\pi_r(i)$ into π_c to form a new individual π'; (3) If π' is better than π_c, replace π_c with π'. Repeat (1) to (3) until all jobs in $\pi_r(i)$ are considered. In the RLS, the job positions of the current sequence π_c are assumed not be proper with respect to the reference sequence. The RLS algorithm will urge jobs to find better positions to be inserted in the current sequence π_c. However, this local search has its own shortcoming. If the reference sequence π_r is fixed, then the searching path is deterministic. In other words, the moving trajectories of all jobs are known. This case would cause the RLS to lack essential self-disturbance ability. If the reference sequence is not changed for many iterations, especially for later period of evolution, the population will be trapped in local optimum. Therefore, a shuffle-based referenced local search (SRLS) is proposed and embedded into DIWO for intensifying exploitation. In this local search, the first searching depends on the best solution so far. In the

Algorithm 2. Random-insertion-based spatial dispersal

```
Input: $\boldsymbol{POP}$, $S_{\min}$, $S_{\max}$, $\sigma_{\min}$ and $\sigma_{\max}$
Output: $\boldsymbol{POP}'$
01: for $i = 1$ to $|\boldsymbol{POP}|$ do
02:     Calculate $S_i$ for $\pi_i$ in $\boldsymbol{POP}$ according to Eq. (10);
03: end for
04: for $i = 1$ to $|\boldsymbol{POP}|$ do
05:     Calculate the $\sigma_{i,k}$ according to Eq. (12);
06:     for $j = 1$ to $S_i$ do
07:         Generate a value of $d_i$ according to Eq. (13), and then check the legality of $d_i$ according to Eq. (14);
08:         $\pi_R \leftarrow \emptyset$ and $\pi' \leftarrow \emptyset$;
09:         $\pi_R \leftarrow$ Randomly remove $d_i$ jobs from $\pi_i$;
10:         $\pi' \leftarrow \pi_i \backslash \pi_R$;
11:         $\pi_R \leftarrow$ Sort all jobs in $\pi_R$ with the non-decreasing sums of their processing time;
12:         for $k = 1$ to $d_i$ do
13:             Test the $k$-th job of $\pi_R$ into all possible positions of $\pi'$;
14:             Insert it into $\pi'$ at the position resulting in the lowest makespan;
15:         end for
16:         $\boldsymbol{POP}' \leftarrow \boldsymbol{POP}' \cup \{\pi'\}$;
17:     end for
18: end for
```

following search, the referenced sequence is shuffled, and the searching path is redefined. The procedure of SRLS is given in Algorithm 3.

In the local search above, shuffle(π_r) is a function to disturb the order of the jobs in π_r. The Fisher–Yates shuffle method (https://en.wikipedia.org/wiki/Fisher-Yates_shuffle) is used as the shuffle function. The variable $flag$ ensures two different searching paths are performed in once local search. One is based on the best individual so far, the other is based on a random sequence. To avoid cycling search and getting trapped in local optimum, we perform the local search for each new seed created in the spatial dispersal procedure with a probability pls. For each new seed, a uniformly random number is generated in the interval [0, 1], if the random number is less than pls, the local search is performed on this new seed.

From the procedure of local search, it can be seen that the computational complexity is $O(2mn^3)$ in the worst case. The computational burden of SRLS is to evaluate each possible inserting position in lines 08–14 of Algorithm 3. Therefore, to reduce the computational time for evaluating them, two accelerating algorithms, i.e., the same speedup method used in PF-NEH(x) and the pruning procedure (Wang et al., 2012) are combined. To be specific, the speedup method is firstly employed to calculate the departure time $d'_{i,j}$ and the tail time $f'_{i,j}$ of job i on machine j in π'. π' is the partial sequence by removing the job $\pi(a)$ from π. The tail time $f'_{i,j}$ is the duration between the latest starting time of job i on machine j and the end of the operation. Then, insert the job $\pi(a)$ into kth position of π' to form a new individual π'' and calculate the departure time $d''_{k,j}$ of $\pi(a)$ on each machine. Thereby, the makespan of π'' can be calculated by:

$$C_{\max}(\pi'') = \max_{j=1,2,\ldots,m} (d''_{k,j} + f'_{k,j}) \tag{15}$$

As regard the pruning procedure, when calculating the maximum value of $d''_{k,j} + f'_{k,j}$, if one of $d''_{k,j} + f'_{k,j}$ has been larger than the makespan of π, π'' must be not superior to π. So it is unnecessary to further evaluate π''. π'' should be discarded and we can begin to consider the next insertion position.

According to the speedup method, the computational complexity to evaluate an insertion operation can be reduced from $O(mn)$ to $O(m)$. The pruning procedure further reduces the computational effort. Thus, the computational complexity of the local search phase can be reduced to $O(2mn^2)$.

4.5. Competitive exclusion

The competitive exclusion determines which weeds can be allowed to pass to the next generation. In the basic IWO, all new created seeds and their parents are ranked together as a colony of weeds with respect to their fitness. Afterwards, weeds with lower fitness are eliminated to reach the maximum allowable population in a colony. However, this mechanism only considers the quality for the next plant population, and the diversity is ignored. If only select the best individuals for next iteration, the evolution of the population would stagnate soon. Therefore, a competitive exclusion proposed by Shao et al. (2017) is employed, in which a certain number of outstanding different individuals are selected as the population for next iteration. Firstly, a simple formula to check the similarity between two individuals is given as follows:

$$\text{distance}\left(\pi_1, \pi_2\right) = \sum_{i=1}^{n} x_i(\pi_1(i), \pi_2(i)) \tag{16}$$

$$x_i\left(\pi_1(i), \pi_2(i)\right) = \begin{cases} 0 & \pi_1(i) = \pi_2(i) \\ 1 & \pi_1(i) \neq \pi_2(i) \end{cases} \tag{17}$$

It can be seen from Eq. (16), π_1 is same as π_2 when distance $\left(\pi_1, \pi_2\right) = 0$. Then, the new created seeds are combined with their parents to form a new plant population. The weeds in this new population are ranked according to their fitness, and then the former $P_{\max}$ outstanding weeds are reserved for next iteration. These reserved individuals are not similar to each other. The detailed implementation of the employed competitive exclusion is given in Algorithm 4.

It should be noted that the size of the initial plant population N_0 actually has less impact upon the final results of the algorithm since the number of weeds in the colony will quickly get the allowable maximum number of plants $P_{\max}$ (may be one iteration when the gap between $P_{\max}$ and N_0 is not large). For simplicity, we set N_0 to $P_{\max}$, which also reduces the number of parameters.

4.6. Overview of the DIWO algorithm

Having described each component of the proposed discrete invasive weed optimization algorithm, we summarize the procedure of it in Algorithm 5. In this algorithm, the PF-NEH(x)-based initialization

Algorithm 3. Shuffle-based referenced local search

```
Input: the individual $\pi$ and the best individual $\pi_r = [\pi_r(1), \pi_r(2), \dots, \pi_r(n)]$ so far
Output: $\pi$
01: $cnt \leftarrow 1$, $j \leftarrow 0$, and $flag \leftarrow 0$;
02: Repeat
03:     while $cnt < n$ do
04:         $j \leftarrow \mathrm{mod}(j+1, n)$;
05:         $\pi' \leftarrow$ Remove the job $\pi_r(j)$ from $\pi$;
07:         $bestfit \leftarrow \infty$;
08:         for $i = 1$ to $n$ do
09:             $\pi'' \leftarrow$ Insert the job $\pi_r(j)$ into the position $i$ of $\pi'$;
10:             if $C_{\max}(\pi'') < bestfit$, then
11:                 $keypos \leftarrow i$;
12:                 $bestfit \leftarrow C_{\max}(\pi'')$;
13:             end if
14:         end for
15:         if $C_{\max}(\pi) > bestfit$, then
16:             $\pi \leftarrow$ Insert job $\pi_r(j)$ into the $keypos$-th position of $\pi'$;
17:             $cnt \leftarrow 1$;
18:         else
19:             $cnt \leftarrow cnt + 1$;
20:         end if
21:     end while
22:     if $flag == 0$, then shuffle($\pi_r$), and $flag \leftarrow flag + 1$;
23: Until $flag == 2$
```

Algorithm 4. Competitive exclusion

```
Input: $\boldsymbol{POP}$, $\boldsymbol{POP}'$ and $P_{\max}$
Output: $\boldsymbol{POP}$
01: $\boldsymbol{POP}'' \leftarrow \emptyset$;
02: $\boldsymbol{POP}'' \leftarrow \boldsymbol{POP} \cup \boldsymbol{POP}'$;
03: $\boldsymbol{POP}'' \leftarrow$Sort the weeds in $\boldsymbol{POP}''$ with the increase of makespan;
04: $\boldsymbol{POP} \leftarrow \emptyset$;
05: $j \leftarrow 2$ and $\boldsymbol{POP} \leftarrow \{\pi_1''\}$, where $\pi_1'' \in \boldsymbol{POP}''$;
06: while $|\boldsymbol{POP}| < P_{\max}$ and $j \le |\boldsymbol{POP}''|$ do
07:     $flag \leftarrow true$;
08:     for $i = 1$ to $|\boldsymbol{POP}|$ do
09:         if distance$(\pi_j'', \pi_i) == 0$, where $\pi_j'' \in \boldsymbol{POP}''$ and $\pi_i \in \boldsymbol{POP}$, then
10:             $flag \leftarrow false$;
11:         end if
12:     end for
13:     if $flag == true$, then $\boldsymbol{POP} \leftarrow \boldsymbol{POP} \cup \{\pi_j''\}$;
14:     $j \leftarrow j + 1$;
15: end while
```

method is employed to provide more promising candidate solutions. The random-insertion-based spatial dispersal is responsible for maintain the searching ability, the shuffle-based referenced local search is used to enhance the exploitation ability, and the competitive exclusion is used for constructing a good population for next iteration. These strategies enable DIWO to converge faster and provide high quality solutions.

5. Numerical experiments

In this section, an extensive experimental evaluation is carried out to investigate the performance of the presented algorithm. First of all, the test instances and the experiment settings are presented in Section 5.1. Then, the experimental results are reported according to the following aspects:

1. Calibration of parameters.

2. Assessment of the effectiveness of the proposed spatial dispersal.

3. Assessment of the effectiveness of the local search.

4. Comparison of the proposed algorithm with the existing well-performing algorithms.

Algorithm 5. DIWO algorithm

```
Input: $P_{\max}$, $S_{\min}$, $S_{\max}$, $\sigma_{\min}$, $\sigma_{\max}$ and $pls$.
Output: The best one in $\boldsymbol{POP}$
01: Initialize the plant population $\boldsymbol{POP}$ (described in Algorithm 1);
02: $k \leftarrow 1$;
03: repeat
        %Reproduction (described lines 01-03 in Algorithm 2)
04:     Calculate the number of seeds generated by each weed in $\boldsymbol{POP}$;
05:     $\boldsymbol{POP'} \leftarrow \emptyset$;
06:     for $i = 1$ to $P_{\max}$ do          %Spatial dispersal (described in lines 04-21 Algorithm 2)
07:         Calculate $\sigma_{i,k}$ for the $i$-th weed of $\boldsymbol{POP}$;
08:         Generate $S_i$ new seeds for the $i$-th weed of $\boldsymbol{POP}$;
09:         Save these $S_i$ new seeds into $\boldsymbol{POP'}$;
10:     end for
11:     for $i = 1$ to $|\boldsymbol{POP'}|$ do          %Local search (described in Algorithm 3)
12:         if $rand < pls$, then
                Perform SRLS on the $i$-th individual in $\boldsymbol{POP'}$;
13:         end if
14:     end for
        %Competitive exclusion (described in Algorithm 4)
15:     Select $P_{\max}$ different outstanding individuals from $\boldsymbol{POP}$ and $\boldsymbol{POP'}$ to construct a new plant population $\boldsymbol{POP}$
        for next iteration;
17:     $k \leftarrow k + 1$;
16: until the termination criterion is met
```

5.1. Experiment settings

To investigate the performance of the DIWO, the benchmark set introduced by Taillard (1993) is adopted. The Taillard's benchmark set consists of 120 instances, ranging from 20 jobs and 5 machines to 500 jobs and 20 machines. These instances are divided into 12 subsets, each of which consists of 10 instances with the same size. Since the BFSP does not have a tight lower bound which is used to compare the results obtained by several algorithms, we adopt best known solutions instead of the lower bound values. Note that calibrating algorithms with the same instances that will later be used for computational results and comparisons constitutes poor practices, which would lead to biased or over-fitting results (Pan and Ruiz, 2014). Therefore, to avoid over-fitting in the results, we randomly regenerate a new test set with $n \in \{40, 80, 120, 160, 200\}$ and $m \in \{5, 10, 15, 20, 25\}$ through Taillard's generation method (Taillard, 1993). Each combination of n and m has five instance, resulting in a total of 125 instances. 125 instances are enough for calibration. The processing time is uniformly distributed from 1 to 99. These test instances are employed to calibrate the parameters of the proposed algorithm.

In this study, all algorithms are coded in Java language and executed on a personal computer that has an Intel Core (TM) CPU with 3.20 GHz, 4G RAM, and Windows 7 Operating System. The termination condition is set as the maximum elapsed CPU time $t = \rho$ mn milliseconds, where ρ is set to different values in the following sections. To analyze the experimental results obtained, the Average Relative Percent Deviation (ARPD) is collected to measure the average relative quality of the solutions, which can be calculated as follows:

$$\text{ARPD} = \frac{1}{R}\sum_{i=1}^{R}\frac{C_i - C_{\min}}{C_{\min}} \times 100 \tag{18}$$

where R is the number of runs. C_i is the solution generated by a specific algorithm in the ith experiment for a given instance. As mentioned above, two different types of instance are employed in this study. The first one is the Taillard's instances, which are used for the comparison of algorithms in the literature. Ribas et al. (2011) have reported the best known solutions (makespan) for these instances, so we set them as $C_{\min}$. The second one is the 125 new instances generated by ourselves, which are used for parameters calibration. These instances do not have the best known solutions. Thus, the minimum makespan found by all algorithms is selected as $C_{\min}$. Obviously, the smaller the value of ARPD is, the better the performance of the algorithm is.

5.2. Design of experiments for the calibration of parameters

The selection of parameters has a significant influence on the effectiveness and efficiency of stochastic algorithms. Usually, the parameters of an algorithm are determined through two ways: the experience in the previous literature and experimental way. For the first way, we can set parameters through summarizing previous relevant literature. Especially for similar problems, the values of parameters in the previous literature have a strong instructive significance, which can further help us narrow the scope of the selection. However, this way often cannot provide accurate values. For the second way, it obtains the optimal calibration through testing many potential values of parameters. Meanwhile, it also analyzes the sensitivity of parameters by using several statistical methods. But it often consumes considerable computational time due to test all combinations of parameters, and its accuracy is influenced by the potential values of parameters. Therefore, in our parameters calibration, the merits of these two ways are combined. To be specific, firstly, the selection range of each parameter for DIWO is determined according to the previous literature (Sang et al., 2018a,b; Shao et al., 2017) and some preliminary experiments. Several levels (values) for a factor (parameter) are determined by trial and error. Then, in order to find the best levels for these factors, the design of experiments (DOE) approach (Montgomery, 2008) is employed. Each combination would be tested by using some additional instances. The multi-factor analysis of variance (ANOVA) procedure would be used to analyze the obtained experimental results to determine the main effects

Table 1
ANOVA results over calibrating the parameters of DIWO.

Source	Sum of squares	Degrees of freedom	Mean Square	F-ratio	p-value
$P_{\max}$	91.01	2	45.5038	321.42	**0.0000**
$S_{\min}$	45.04	2	22.5224	159.09	**0.0000**
$S_{\max}$	2.71	2	1.3554	9.57	**0.0001**
$\sigma_{\min}$	11.80	2	5.9012	41.68	**0.0000**
$\sigma_{\max}$	0.21	2	0.1055	0.75	0.4746
pls	9.88	2	4.9415	34.90	**0.0000**
$P_{\max} * S_{\min}$	33.96	4	8.4898	59.97	**0.0000**
$P_{\max} * S_{\max}$	2.50	4	0.6250	4.41	**0.0014**
$P_{\max} * \sigma_{\min}$	0.78	4	0.1959	1.38	0.2368
$P_{\max} * \sigma_{\max}$	0.87	4	0.2168	1.53	0.1899
$P_{\max} * pls$	22.56	4	6.2901	44.43	**0.0000**
$S_{\min} * S_{\max}$	2.28	4	0.5697	4.02	**0.0029**
$S_{\min} * \sigma_{\min}$	0.66	4	0.1661	1.17	0.3204
$S_{\min} * \sigma_{\max}$	1.22	4	0.3042	2.15	0.0720
$S_{\min} * pls$	6.20	4	1.5494	10.94	**0.0000**
$S_{\max} * \sigma_{\min}$	10.87	4	2.7184	19.20	**0.0000**
$S_{\max} * \sigma_{\max}$	0.62	4	0.1557	1.10	0.3548
$S_{\max} * pls$	0.30	4	0.0761	0.54	0.7082
$\sigma_{\min} * \sigma_{\max}$	0.79	4	0.1964	1.39	0.2355
$\sigma_{\min} * pls$	0.37	4	0.0932	0.66	0.6209
$\sigma_{\max} * pls$	0.18	4	0.0459	0.32	0.8620
Residual	8246.11	58 247	0.1416		
Total	8493.54	58 319			

and interaction of parameters, and then establish the best combination of parameters.

From the procedure of the DIWO algorithm, there are six key parameters: the allowable maximum number of plants ($P_{\max}$), the minimum number of seeds ($S_{\min}$), the maximum number of seeds ($S_{\max}$), the minimum value of standard deviation ($\sigma_{\min}$), the maximum value of standard deviation ($\sigma_{\max}$) and the probability of the local search (pls). The levels for each factor are set as follows: $P_{\max} \in \{10, 30, 50\}$, $S_{\min} \in \{0, 1, 2\}$, $S_{\max} \in \{5, 7, 9\}$, $\sigma_{\min} \in \{3, 4, 5\}$, $\sigma_{\max} \in \{8, 9, 10\}$, and $pls \in \{0.15, 0.25, 0.45\}$. These parameters above yield a total of $3\times3\times3\times3\times3\times3 = 729$ different configurations for the DIWO algorithm. Then, we employ DOE approach to study the influence of parameters on the performance of the DIWO algorithm. Each configuration is executed 5 times for each instance. The termination criterion is set $\rho = 30$. As a result, a total of $729 \times 5 \times 125 = 455\,625$ results are generated in this experiment. The total CPU time for calibration is almost 284.76 CPU days. Due to multi-cores in our personal computer, actually we only used almost 9.5 days to complete the whole experiment.

The experimental results are analyzed by means of ANOVA. In the experiment, three main hypotheses, i.e., normality, homogeneity of variance and independent of the residuals, are checked and accepted. Note that we focus on the F-ratio which is a clear indicator of significance when the p-values are less than the confidence level. The larger the F-ratio is, the more effect the factor has on the response variable. Moreover, we do not consider the interactions more than two factors since their F-ratios are very small. The ANOVA results are reported in Table 1.

As seen in Table 1, the p-values of parameter $P_{\max}$, $\sigma_{\min}$, $S_{\min}$, pls and $S_{\max}$ are less than $\alpha = 0.05$ confidence level, which indicates that these factors are significant and influence the performance of the DIWO algorithm. The allowable maximum number of plants ($P_{\max}$) achieves the largest F-ratio, which indicates that $P_{\max}$ is the significant factor that affects the performance of the proposed DIWO. Fig. 2 gives the main effects plot of all parameters. It is clearly observed that the choice of $P_{\max} = 10$ obtains the best result, while $P_{\max} = 30$ yields the worst result. It demonstrates that a larger population increases the computational time in once iteration and reduce the convergence speed which are not desirable. The second largest F-ratio value corresponds to the factor $S_{\min}$. It can be observed from Fig. 2 that the value $S_{\min} = 0$ yields the best result, while $S_{\min} = 2$ obtains the worst result. It can be explained that a larger $S_{\min}$ may enlarge the number of offspring individuals with lower fitness in each generation, which would decrease the quality of the whole population. The next greatest F-ratio corresponds to pls. It can be seen that $pls = 0.45$ or 0.25 provides the worse results, while $pls = 0.15$ obtains the best result. It demonstrates that the larger probability would lead the population to trap into local optimum. The fourth F-ratio value is relative to $\sigma_{\min}$. As seen in Fig. 2, $\sigma_{\min} = 5$ achieves the best result, while $\sigma_{\min} = 3$ yields the worst result. Since $\sigma_{\min}$ determines the smallest searching range, a lower value of $\sigma_{\min}$ would lead to inadequate searching. The last significant factor is $S_{\max}$. It is clear from Fig. 2 that too small or too large $S_{\max}$ would lead to deterioration of the algorithm's performance, while $S_{\max} = 7$ gives the best average performance.

However, if there are significant interactions between parameters, it is not meaningful to consider the value of a single parameter. Table 1 also displays the 2-level interactions of all parameters. It is observed from it that the interactions $P_{\max} * S_{\min}$, $P_{\max} * S_{\max}$, $P_{\max} * pls$, $S_{\max} * S_{\min}$, $S_{\min} * \sigma_{\min}$ and $S_{\min} * pls$ are significant since the p-values are less than 0.05. Thus, we have to consider the interactions of these parameters. Fig. 3 shows the interaction plot of these parameters. Although many interactions are statistically significant, all interactions except for $S_{\max} * S_{\min}$ are weak according to Fig. 3 and they do not influence the conclusions reached from Fig. 2. For the interaction $S_{\max} * S_{\min}$, $S_{\min} = 0$ generates the lowest ARPD with $S_{\max} = 5$, which is different from Fig. 2. Nonetheless, from other interactions related to $S_{\max}$, we can see that $S_{\max} = 7$ yields the good results. Moreover, the F-ratio of $S_{\max} * S_{\min}$ is relatively smaller than other factors. Thus, the interaction $S_{\max} * S_{\min}$ is also weak, and $S_{\max}$ should be set to 7.

However, although Fig. 2 has shown the best value for each parameter, we cannot confirm that whether lower values for $P_{\max}$ and $S_{\max}$, and larger values for $\sigma_{\min}$ can obtain better performance. Therefore, an additional experiment is conducted to find the best values for them. In this experiment, we set $P_{\max} \leq 10$, since 10 is the current best value for $P_{\max}$ in the previous experiment. The potential values of $P_{\max}$ are set {4, 6, 8, 10}. Similarly, we set $pls \leq 0.15$, i.e. $pls \in \{0.05, 0.10, 0.15\}$. Moreover, since $5 \leq \sigma_{\min} \leq \sigma_{\max}$ and the algorithm achieves the best performance when $\sigma_{\max} = 10$, the potential values of $\sigma_{\min}$ are included in $\{5, 7, 9, 10\}$. For other three parameters, i.e., $S_{\max}$, $S_{\min}$ and $\sigma_{\max}$, we set them according to Fig. 2. Thus, there are a total of $4\times3\times4 = 48$ different configurations. The new 125 instances generated in Section 5.1 are also used as the test bed. Each configuration is run 5 times for each instance. The termination criterion is still set $\rho = 30$. As a result, a total of $48 \times 5 \times 125 = 30\,000$ results are generated in this experiment. The results above are also analyzed by the ANOVA technique. The analysis results are reported here in the form of ANOVA charts with 95% confidence intervals in Fig. 4. Note that overlapping confidence intervals signify that the observed differences in the response variables are not statistically significant. In other words, if the confidence intervals of two potential values for a parameter are overlapped, these two potential values cannot bring the significant changes for the algorithm.

As can be seen from Fig. 4, when $P_{\max} = 10$ and $S_{\max} = 0.15$, DIWO can achieve the best performance, which is in consistent with Fig. 2. Meanwhile, the confidence intervals of $P_{\max} = 10$ and $S_{\max} = 0.15$ are not overlapped with other potential values, which indicates $P_{\max} = 10$ and $S_{\max} = 0.15$ are the best selections for DIWO. Additionally, we can also conclude that lower $P_{\max}$ and $S_{\max}$ cannot enable DIWO to obtain better performance. However, $\sigma_{\min}$ has a different behavior. It can be seen from Fig. 4 that the performance of DIWO on $\sigma_{\min} = 7$ and $\sigma_{\min} = 10$ is better than on $\sigma_{\min} = 7$ and $\sigma_{\min} = 9$. But these differences are not significant, since the confidence intervals of these potential values are overlapped. In other words, the larger $\sigma_{\min}$ cannot bring the significant enhancement for DIWO, and $\sigma_{\min}$ can be set to a value between 5 and 10.

On the basis of the analysis above, the parameters are recommended as follows: $P_{\max} = 10$, $S_{\min} = 0$, $S_{\max} = 7$, $\sigma_{\min} = 0$, $\sigma_{\max} = 5$, and $pls = 0.15$.

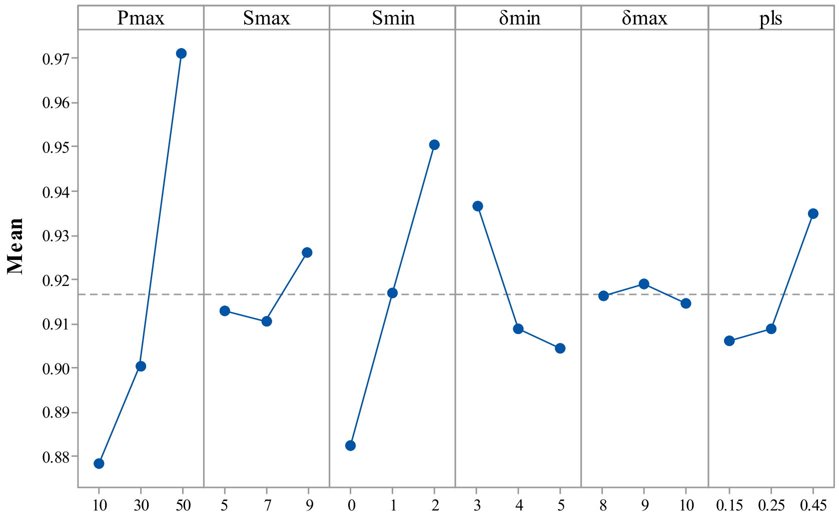

Fig. 2. Main effects plot of parameters.

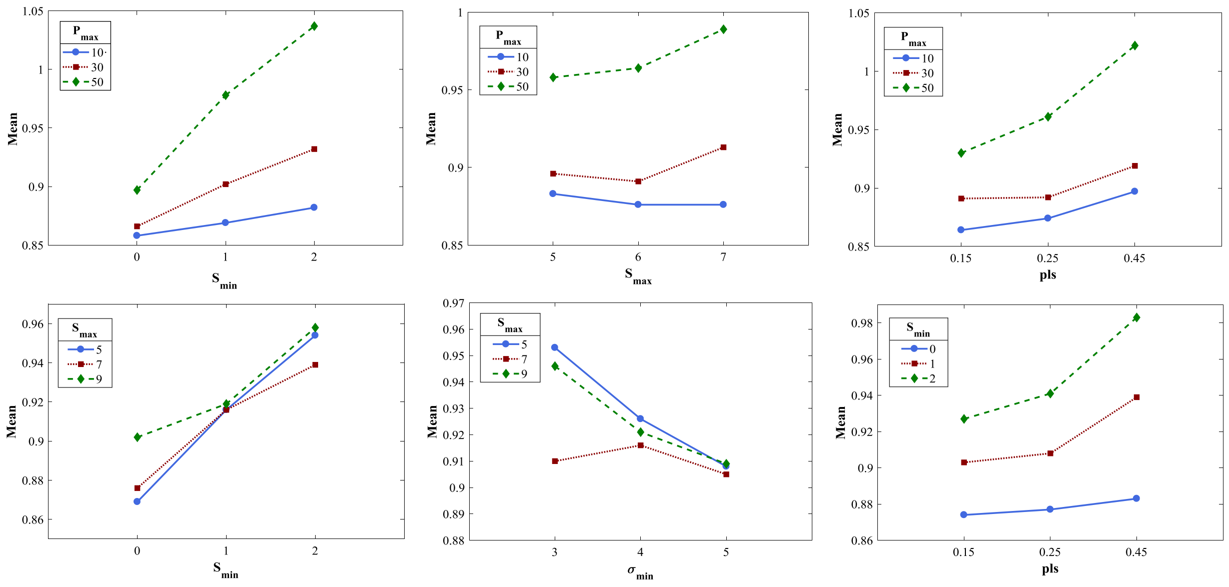

Fig. 3. Interaction plot of parameters.

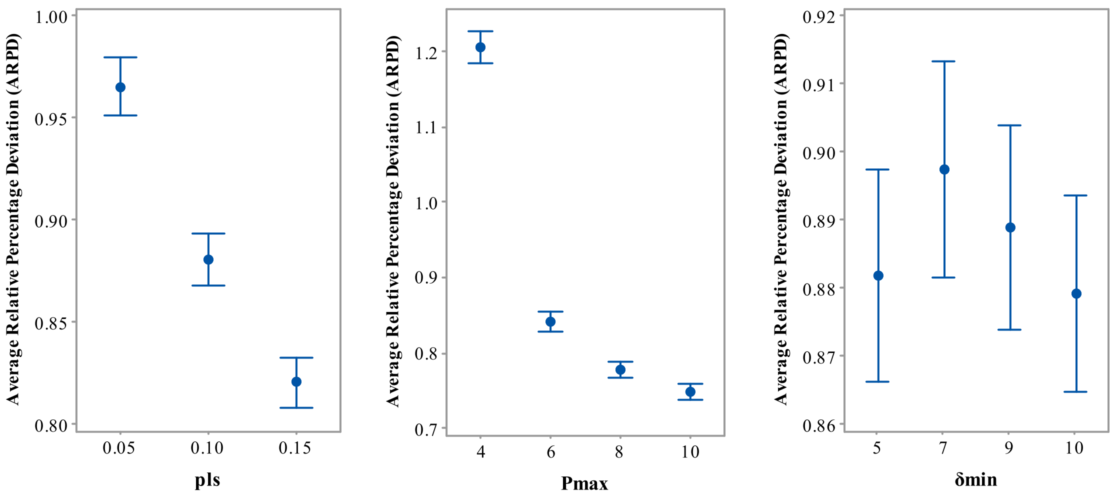

Fig. 4. Means plot and 95% confidence level intervals of P_{max}, S_{max} and σ_{min}.

Table 2
The ARPD of SIWO_NL, SDIWO, DIWO_NL, SaDIWO_NS, SaDIWO and DIWO.

Instance	SDIWO_NL	SDIWO	DIWO_NL	SaDIWO_NS	SaDIWO	DIWO
20 × 5	2.396	0.008	0.078	0.012	0.017	**0.000**
20 × 10	2.556	0.007	0.046	0.013	0.008	**0.000**
20 × 20	1.727	**0.000**	0.019	0.010	0.010	**0.000**
50 × 5	3.351	0.450	0.374	0.028	0.021	**0.128**
50 × 10	3.572	0.228	0.224	0.072	0.105	**−0.073**
50 × 20	5.090	0.231	0.521	0.248	0.312	**0.086**
100 × 5	1.405	−0.218	−0.587	−0.776	−0.746	**−0.775**
100 × 10	1.764	−0.582	−0.652	−0.672	−0.657	**−0.972**
100 × 20	2.922	−0.336	−0.173	−0.119	−0.139	**−0.741**
200 × 10	0.452	−0.809	−1.021	−0.759	−0.763	**−1.200**
200 × 20	0.500	−1.245	−1.296	−0.917	−0.889	**−1.591**
500 × 20	−1.682	−2.594	−2.787	−2.438	−2.463	**−2.835**
Average	2.005	−0.405	−0.438	−0.441	−0.432	**−0.664**

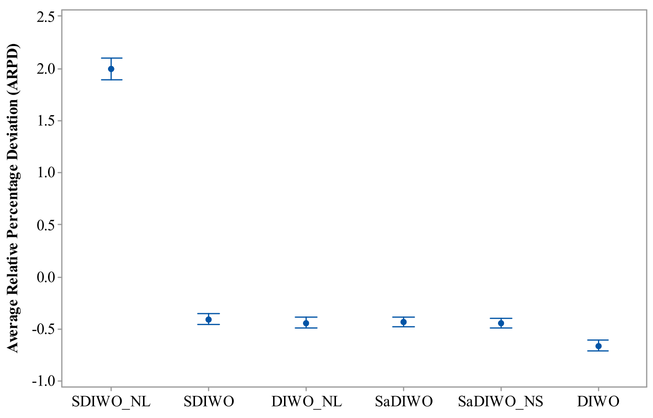

Fig. 5. Means plots and 95% confidence level intervals of SDIWO_NL, SDIWO, DIWO_NL, SaDIWO_NS, SaDIWO and DIWO.

5.3. *Performance of the proposed spatial dispersal model*

The spatial dispersal is important for IWO, which determines the main searching ability of IWO. To show the performance of the proposed spatial dispersal, several computational experiments are designed as follows. Due to no other variants of IWO directly used to solve the BFSP, an effect discrete IWO (named SDIWO for short in this paper) proposed by Sang et al. (2018b) to solve the lot-streaming flow-shop scheduling problems is implemented in the first trail. The SDIWO algorithm adopts a spatial dispersal based on the distance of random insertion. To make a fair comparison, the same initial method with our proposed DIWO is used in SDIWO. In the second trial, our proposed DIWO algorithm is executed. Generally, the local search has important influence on the performance of the main algorithm. In order to purely demonstrate the searching ability of the proposed spatial dispersal, in the third and fourth trial, SDIWO without any local searches (named SDIWO_NL) and DIWO without any local searches (DIWO_NL) are executed, respectively. Through comparing the experimental results of these four trials, the contribution of the proposed spatial dispersal can be clearly ascertained. Moreover, Shao et al. (2017) proposed a self-adaptive discrete IWO (SaDIWO) to solve the BFSP with minimizing total tardiness, which has similar structure with our proposed DIWO. In the five trial, the spatial dispersal employed by SaDIWO is replaced with our proposed one, and a variant named SaDIWO_NS is developed. Meanwhile, we also adapt and implement SaDIWO to solve the considered BFSP. Based the comparison of SaDIWO_NS, SaDIWO, DIWO, we can further confirm the performance and applicability of our proposed spatial dispersal, and the worse performance of SaDIWO is whether attributed to its original spatial dispersal. The parameters of SDIWO and SaDIWO are consistent with its original paper. We run these compared algorithms for $\rho = 30$, that is $t = 30$ mn. Each instance is independently run 10 times. Computational results are given in Table 2.

As can be seen from Table 2, DIWO achieves a smallest average ARPD of −0.664. The SDIWO_NL yields the worst results among all compared algorithms. A 95% confidence interval plot is presented in Fig. 5 to show the performance of these algorithms. It can be clearly observed from this figure that DIWO outperforms other compared algorithms from a statistical point. Note that overlapping intervals denote statistically insignificant differences between the plotted overlapped means. So we can see from Fig. 5 that the DIWO_NL has the same performance with SDIWO since their confidence intervals almost coincide, whereas its results are worse than DIWO. It indicates that although DIWO_NL does not have local search to enhance its performance, it still has a very strong searching ability, which is because the proposed spatial dispersal stresses the balance between global exploration and local exploitation. In the proposed spatial dispersal, the better weeds in population have small values of d (i.e., the number of removed jobs), while the worse weeds have large values of d. The better weeds are used to generate small neighborhood to exploit the solution space, while the worse weeds are used to generate large neighborhood to explore the solution space. As also seen in Fig. 5, SDIWO_NL is much worse than SDIWO, which means that SDIWO_NL has poor searching ability. The performance of SDIWO fully depends on the local search. The spatial dispersal in SDIWO may only play a role of global exploration for solution space. Moreover, as can be seen in Table 4, SaDIWO is slightly inferior to SaDIWO_NS, which indicates that our proposed spatial dispersal can enhance the performance of SaDIWO but the enhancement is limited. It also reveals that the spatial dispersal is not the main reason for the worse performance of SaDIWO. Meanwhile, Fig. 5 shows that the differences between SaDIWO and SaDIWO_NL are not statistically significant, which further confirms the conclusions above. Therefore, the results above demonstrate that the proposed spatial dispersal has good performance and applicability, and can effectively enhance the performance of the algorithm.

5.4. *Performance of the local search*

In this section, we would like to evaluate the benefits of using the proposed local search and the speedup method. For this reason, some experiments are designed as follows. It should be noted that we regard the prune procedure as a part of the speedup method. The first trial is carried out to show the effectiveness of SRLS for the DIWO. The DIWO with SRLS (DIWO), the DIWO with RLS (DIWO_RLS) and the DIWO without any local searches (DIWO_NL), i.e., $pls = 0$, are executed. The second trial is carried out to test that the insertion neighborhood is more suitable than the swap neighborhood for BFSP. The insertion neighborhood in SRLS is replaced by the swap neighborhood. Since the speedup method in this paper is not suitable for evaluating the swap neighboring solutions, we adopt the speedup mechanism proposed by Li et al. (2009), which can save at least 50% computational time. This algorithm is denoted by DIWO_SP. The third trial is carried out to verify the effectiveness of the speedup method for SRLS. In this trial, SRLS without the speedup method (DIWO_NS) is executed. By comparing the results of these experiments, we can confirm the effectiveness of the SRLS and the speedup method. Additionally, SaDIWO mentioned in Section 5.3 also participates in comparison. Meanwhile, a variant of SaDIWO (denoted as SaDIWO_SRLS) is developed in the last trial, which replaces the local search (variable neighborhood search) of SaDIWO with SRLS. Through comparing SaDIWO_SRLS, SaDIWO and DIWO, we can further confirm the applicability and effectiveness of our proposed local search procedure, and the worse performance of SaDIWO is whether due to its own local search strategy. The termination criterion is also set $\rho = 30$, that is $t = 30$ mn. For each instance, all compared methods are run 10 times independently. The computational results are reported in Table 3.

As revealed in Table 3, the DIWO achieves the smallest ARPD with −0.664 compared to the corresponding values of −0.438, −0.142,

Table 3
The ARPD of DIWO_NL, DIWO_SP, DIWO_NS, DIWO_RLS, SaDIWO_SRLS, SaDIWO and DIWO.

Instance	DIWO_NL	DIWO_SP	DIWO_NS	DIWO_RLS	SaDIWO_SRLS	SaDIWO	DIWO
20 × 5	0.078	0.068	0.049	0.045	0.038	0.017	**0.000**
20 × 10	0.046	0.049	0.003	0.002	0.015	0.008	**0.000**
20 × 20	0.019	0.018	0.001	0.000	0.005	0.010	**0.000**
50 × 5	0.374	0.303	0.566	0.152	0.305	0.021	**0.128**
50 × 10	0.224	0.276	0.352	0.021	0.100	0.105	**−0.073**
50 × 20	0.521	0.683	0.571	0.162	0.239	0.312	**0.086**
100 × 5	−0.587	−0.440	0.171	−0.709	−0.622	−0.746	**−0.775**
100 × 10	−0.652	−0.304	0.079	−0.916	−0.840	−0.657	**−0.972**
100 × 20	−0.173	0.463	0.616	−0.612	−0.589	−0.139	**−0.741**
200 × 10	−1.021	−0.469	−0.262	−1.143	−0.975	−0.763	**−1.200**
200 × 20	−1.296	−0.284	−0.431	−1.477	−1.464	−0.889	**−1.591**
500 × 20	−2.787	−2.070	−2.320	−2.777	−2.698	−2.463	**−2.835**
Average	−0.438	−0.142	−0.050	−0.604	−0.540	−0.432	**−0.664**

−0.050, −0.604, −0.540, −0.432 obtained by DIWO_NL, DIWO_SP, DIWO_NS, SaDIWO_SRLS, SaDIWO, and DIWO_RLS, respectively. It is obvious that the DIWO outperforms other compared algorithms. Fig. 6 displays the means plots and 95% confidence level intervals of all compared algorithms. As seen in this figure, DIWO_NS performs very poor by comparing with other compared algorithms. This is because whether in the spatial dispersal or in the local search, a certain amount of partial sequences will be evaluated, which consumes considerable computational time so that all mechanisms of DIWO almost lose its efficacy. It can be also seen from Fig. 6 that DIWO_SP is inferior to DIWO_RLS and DIWO, which indicates that the insertion neighborhood is more effective than the swap neighborhood. Moreover, we also compare the effectiveness of RLS and SRLS for DIWO. As seen from Fig. 6, the confidence intervals of DIWO_RLS and DIWO do not completely overlap with each other, which indicates that SRLS is more suitable for DIWO than RLS. Meanwhile, we can clearly observe in Fig. 6 that DIWO is better than DIWO_NL, which reveals the proposed local search can effectively improve the performance of DIWO. Additionally, as seen from Table 3, SaDIWO is worse than DIWO and SaDIWO_SRLS, which indicates the proposed SRLS can enhance the performance of algorithm. Meanwhile, it also reveals that the worse performance of SaDIWO is attributed to the local search procedure. In the SaDIWO, an exhaustive variable neighborhood search is embedded, which consumes much time in one iteration, and largely influences the algorithm's ability to perform more iterations to explore more promising solutions. Our proposed SRLS is a lightweight local search mechanism, which cannot only ensure sufficient exploitation for local area, but also does not influence the exploration ability of algorithm. It can be also observed from Fig. 6 that DIWO and SaDIWO_NLS are significantly better than SaDIWO, which further confirms the conclusions above. Therefore, on the basis of the results and analyses above, it can be concluded that SRLS has good performance and applicability, and can effectively enhance the performance of the proposed algorithm.

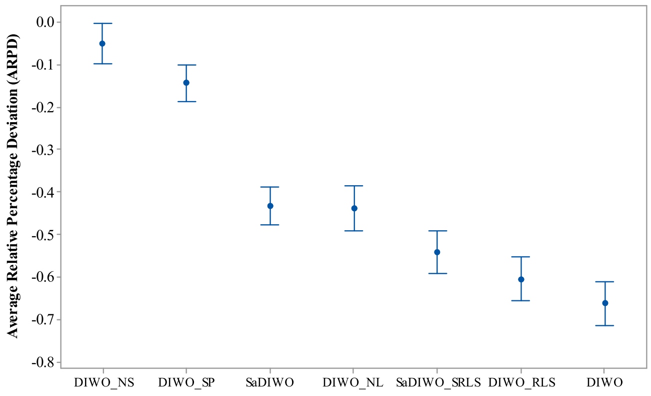

Fig. 6. Means plots and 95% confidence level intervals of DIWO_NL, DIWO_SP, DIWO_NS, DIWO_RLS, SaDIWO_SRLS, SaDIWO and DIWO algorithm.

5.5. *Comparison of other algorithms*

To investigate the effectiveness of the proposed DIWO, we compare it with several state-of-the-art algorithms published in recent literature. These algorithms include HDDE (Wang et al., 2010a), hmgHS (Wang et al., 2011), IG (Ribas et al., 2011), TPA (Wang et al., 2012), RAIS (Lin and Ying, 2013), MA (Pan et al., 2013b), SVNS_D (Ribas et al., 2013), HVNS (Moslehi and Khorasanian, 2014a), DE-ABC (Han et al., 2015b), DE_PLS (Tasgetiren et al., 2015), MFFO (Han et al., 2016a), SaDIWO (Shao et al., 2017), and P-EDA (Shao et al., 2018a). As mentioned from Črepinšek et al. (2014), all experiments should be carried out and compared under the same or stricter conditions. If different algorithms are implemented on different computers, the implementation settings of different algorithms will be different. The running time of each algorithm may be also affected by the operating system and other applications running the experiments. Additionally, these algorithms were often coded by different programming languages and adopted different termination conditions. The reported results in the literature come from different upper bounds. Thus, to make a fair and reasonable comparison, we strictly re-implement all compared algorithms with same programming language and execute them on the same computer environment. We can ensure that all algorithms have same CPU power and time available. All algorithms adopt the same termination criterion $t = \rho$ mn, where ρ is tested at three values 30, 60 and 90 to test how the different algorithms perform with different CPU time. This termination criterion has been increasingly used in recent literature on scheduling (Karabulut, 2016; Pan et al., 2013b; Ribas et al., 2015). Each instance is independently executed 10 replications. It should be noted that TPA is terminated when reaching the maximum number of iterations in its original paper. The performance of TPA largely depends on the maximum number of iterations, so we set it as 1.8×10^6 based on the termination criterion considered. As suggested in Ribas et al. (2013), the parameters of SVNS_D are taken as $\alpha = 0.75$, $\beta = 0.5$ and $ds = 8$. We also use the PF-NEH(x) heuristic to generate the initial solution for SVNS_D. The values of parameters used for other re-implemented algorithms are recommended by their respective original works. Moreover, SaDIWO is initially used to solve BFSP with total tardiness, so we adapt its objective evolution function, and then use the same initialization strategy with the proposed DIWO to generate its initial population. Meanwhile, the adopted speedup method is also incorporated into it. Tables 4 to 6 summarize the ARPD of all compared algorithms for each termination criterion. Best values are printed in **boldface** in these tables.

It is clear from Tables 4 to 6 that the presented DIWO achieves the lowest overall average ARPD for all elapsed CPU time. MA, DE_PLS and SaDIWO are also competitive for some instances. To be specific, for $\rho = 30$ termination criterion (Table 4), only DIWO could always obtain the best solutions on small instances (i.e., 20 jobs). MA has good performance on 50 × 5 and 50 × 20. For large instances (i.e., 200 and 500 jobs), MA, SaDIWO, P-EDA and DIWO are more competitive than

Table 4
The ARPD of the all compared algorithms ($\rho = 30$).

Instance	HDDE	hmgHS	IG	TPA	RAIS	MA	SVNS_D	HVNS	DE-ABC	DE_PLS	MFFO	SaDIWO	P-EDA	DIWO
20 × 5	0.013	0.121	0.065	0.008	0.230	0.005	0.069	0.025	0.014	0.007	0.005	0.017	0.060	**0.000**
20 × 10	0.012	0.057	0.107	0.014	0.079	0.007	0.034	0.004	0.016	0.006	0.007	0.008	0.009	**0.000**
20 × 20	0.001	0.031	0.104	**0.000**	0.017	0.001	0.003	**0.000**	0.009	0.001	0.003	0.010	**0.000**	**0.000**
50 × 5	0.414	0.574	1.220	0.346	0.634	**0.058**	0.196	0.298	1.842	0.229	0.705	0.021	0.155	0.128
50 × 10	0.191	0.398	1.374	−0.045	0.336	−0.039	0.296	0.086	1.463	−0.040	0.395	0.105	0.040	**−0.073**
50 × 20	0.223	0.360	1.429	−0.063	0.978	**0.009**	0.561	0.037	1.210	−0.016	0.342	0.312	0.104	0.086
100 × 5	1.354	0.719	0.929	−0.284	0.062	−0.405	−0.609	−0.351	2.428	0.111	0.916	−0.746	−0.647	**−0.775**
100 × 10	0.596	0.127	1.041	−0.830	0.095	−0.811	−0.465	−0.696	1.568	−0.357	0.135	−0.657	−0.800	**−0.972**
100 × 20	0.403	0.090	1.149	**−0.951**	1.765	−0.780	0.120	−0.734	1.211	−0.387	0.016	−0.139	−0.616	−0.741
200 × 10	2.059	1.727	1.935	−0.789	1.722	−0.737	−0.742	−0.530	2.286	−0.224	1.180	−0.763	−0.939	**−1.200**
200 × 20	0.909	0.614	1.282	−1.480	2.802	−1.343	−0.857	−1.191	1.045	−0.606	−0.028	−0.889	−0.281	**−1.591**
500 × 20	1.171	1.078	1.376	−1.933	4.083	−2.540	−2.425	−0.148	0.768	−2.139	0.038	−2.463	−2.583	**−2.835**
Average	0.612	0.491	1.001	−0.501	1.067	−0.548	−0.318	−0.267	1.155	−0.285	0.310	−0.432	−0.542	**−0.664**

Table 5
The ARPD of the all compared algorithms ($\rho = 60$).

Instance	HDDE	hmgHS	IG	TPA	RAIS	MA	SVNS_D	HVNS	DE-ABC	DE_PLS	MFFO	SaDIWO	P_EDA	DIWO
20 × 5	0.004	0.094	0.031	0.012	0.227	**0.000**	0.050	0.017	0.007	**0.000**	0.002	0.012	**0.000**	**0.000**
20 × 10	0.008	0.039	0.068	0.028	0.080	0.005	0.007	0.002	0.012	0.006	0.003	0.003	**0.000**	**0.000**
20 × 20	0.002	0.014	0.058	**0.000**	0.017	**0.000**	0.008	**0.000**	0.004	**0.000**	**0.000**	0.004	**0.000**	**0.000**
50 × 5	0.237	0.438	1.085	0.317	0.575	**−0.074**	0.147	0.202	1.695	0.062	0.563	−0.066	0.050	0.016
50 × 10	0.069	0.347	1.200	−0.028	0.283	−0.081	0.120	−0.036	1.376	−0.113	0.284	−0.004	−0.037	**−0.115**
50 × 20	0.078	0.312	1.293	−0.018	0.363	**−0.051**	0.497	−0.053	1.115	−0.130	0.232	0.212	0.136	0.000
100 × 5	0.985	0.401	0.699	−0.324	0.048	−0.577	−0.733	−0.554	2.277	−0.210	0.682	**−0.926**	−0.813	−0.809
100 × 10	0.270	−0.086	0.814	−0.831	−0.178	−1.002	−0.638	−0.834	1.495	−0.618	−0.025	−0.868	−0.951	**−1.060**
100 × 20	0.127	−0.161	0.924	**−0.966**	0.036	−0.848	−0.134	−0.911	1.103	−0.636	−0.161	−0.396	−0.695	−0.848
200 × 10	1.816	1.446	1.542	−0.786	0.587	−0.891	−0.850	−0.732	2.190	−0.347	0.810	−0.919	−1.070	**−1.285**
200 × 20	0.645	0.437	0.916	−1.555	0.419	−1.482	−0.986	−1.419	0.961	−0.833	−0.262	−1.118	−1.442	**−1.691**
500 × 20	0.957	0.933	1.270	−2.030	1.454	−2.623	−2.483	−1.883	0.595	−2.230	−0.243	−2.464	−2.669	**−2.925**
Average	0.433	0.351	0.825	−0.515	0.326	−0.635	−0.416	−0.517	1.069	−0.421	0.157	−0.544	−0.624	**−0.726**

Table 6
The ARPD of the all compared algorithms ($\rho = 90$).

Instance	HDDE	hmgHS	IG	TPA	RAIS	MA	SVNS_D	HVNS	DE-ABC	DE_PLS	MFFO	SaDIWO	P-EDA	DIWO
20 × 5	0.001	0.077	0.010	0.013	0.246	0.000	0.036	0.013	0.001	0.003	**0.000**	**0.000**	**0.000**	**0.000**
20 × 10	0.008	0.033	0.050	0.027	0.064	0.003	0.003	**0.000**	0.008	0.002	0.002	**0.000**	**0.000**	**0.000**
20 × 20	**0.000**	0.010	0.033	**0.000**	0.020	0.000	0.001	**0.000**	**0.000**	**0.000**	0.001	**0.000**	**0.000**	**0.000**
50 × 5	0.148	0.365	0.998	0.292	0.556	**−0.111**	0.097	0.149	1.644	−0.062	0.447	−0.168	0.054	−0.080
50 × 10	−0.049	0.265	1.083	−0.036	0.291	−0.141	0.126	−0.067	1.277	**−0.255**	0.218	−0.057	−0.067	−0.173
50 × 20	−0.004	0.275	1.209	−0.032	0.394	−0.034	0.441	−0.073	1.078	**−0.210**	0.222	0.191	0.092	−0.094
100 × 5	0.737	0.211	0.591	−0.340	0.019	−0.836	−0.817	−0.553	2.188	−0.281	0.393	**−1.053**	−0.858	−0.948
100 × 10	0.019	−0.191	0.616	−0.903	−0.243	−1.051	−0.758	−0.879	1.436	−0.823	−0.173	−0.938	−0.107	**−1.068**
100 × 20	−0.024	−0.222	0.783	−0.963	−0.109	−0.958	−0.247	−0.965	1.064	−0.819	−0.230	−0.555	−0.767	**−0.971**
200 × 10	1.670	1.324	1.298	−0.810	0.473	−0.988	−0.904	−0.743	2.121	−0.414	0.630	−1.038	−1.169	**−1.359**
200 × 20	0.532	0.266	0.721	−1.579	0.123	−1.580	−1.110	−1.468	0.934	−0.944	−0.407	−1.262	-1.543	**−1.782**
500 × 20	0.874	0.863	1.041	−2.053	0.936	−2.676	−2.529	−1.976	0.480	−2.245	−0.414	−2.547	−2.710	**−2.970**
Average	0.326	0.273	0.703	−0.532	0.231	−0.698	−0.472	−0.547	1.019	−0.504	0.057	−0.617	−0.633	**−0.787**

other algorithms. For $\rho = 60$ termination criterion, it can be seen from Table 5 that DIWO has good performance on most instances, except for 50 × 5, 50 × 20, 100 × 5 and 100 × 20. For these instances, MA, TPA and SaDIWO yield the good results, respectively. For the small instances (i.e., 20 jobs), P-EDA and DIWO achieves the best results. For the last termination criterion $\rho = 90$, DIWO has good performance on small and large instances, but for instances that have 50 jobs, DIWO is slightly inferior to MA and DE_PLS. Additionally, taken together from all termination criteria, for two variants of variable neighborhood search, HVNS is better than SVNS_D. As a swarm intelligence optimization algorithm, MFFO provides better results than HDDE, hmgHS and IG. For the non-population based algorithms, i.e., TPA, SVNS_D, HVNS, DE_PLS, and IG, TPA obtains the best performance. For another discrete variant of IWO, i.e., SaDIWO, it has similar framework with DIWO, but its reproduction, spatial dispersal and local intensification phases are different from DIWO. Especially for local intensification phase, SaDIWO adopts a variable neighborhood search (VNS) based on insertion, swap, and double-edge insertion. From the results in Tables 4 to 6, SaDIWO is inferior to DIWO. The main reason behind this is that the VNS of SaDIWO consume much time, which diminishes the algorithm's capability to perform more iterations so that the exploration time suffers a great compression. That is to say, over local intensification influences the balance between exploration and exploitation. Another main reason is that the type of the neighborhood structure also influences the performance of algorithm. In Section 5.4, it has been validated that the swap neighborhood structure could not play a decisive role for BFSP with makespan. The swap neighborhood structure consumes a certain search time for SaDIWO. Instead, DIWO employs a single insertion neighborhood structure and controls it with probability way, which effectively implements local intensification and ensures the balance between exploitation and exploration. Additionally, as an evolutionary algorithm based on statistics, P-EDA is also very competitive, but which is still inferior to DIWO. Compared with DIWO, P-EDA is relatively complicated, and needs to construct a probabilistic model for a promising population. The accuracy of the probabilistic model largely influences the performance of P-EDA. DIWO explores the whole solution space only through a simple random insertion mechanism, which makes DIWO has lower complexity. Meanwhile, an effective local search ensures the local exploration ability of DIWO.

Additionally, from the results above, we can see that the compared algorithms have different performance on different scale instances and termination criterion. To investigate these behaviors, Fig. 7 firstly shows the interaction plot between algorithm and the number of jobs (n) with machines (m). It can be seen from this figure that all compared algorithms are more influenced by m and n. In more detail, most algorithms perform better when n increases, except for RAIS, hmgHS, HDDE and IG. For the interaction of algorithm and m, we can observe that IG and RAIS perform worse and worse with the increase of the number of machines, while other algorithms yield the opposite results. Meanwhile, we can also see that the proposed DIWO performs better and better with the increase of n and m, which also reveals DIWO is good at solving the large instances. Moreover, to investigate the behavior of the algorithms on different termination criteria, the interaction between algorithm and ρ is shown in Fig. 8. As seen this figure, the performance of RAIS is rather poor for $\rho = 30$, but it achieves good results for $\rho = 60$ and $\rho = 90$. The RAIS is more influenced by ρ and the reason for this may lie in that no effective local search in it so that it requires more time to get better results. The TPA obtains the similar results for each value of ρ, which means that TPA converges rapidly and substantial additional CPU time is not helpful to enhance its performance. For other algorithms, additional running time can help them obtain better results, but the improvement is very limited. In other words, these algorithms cannot be expect to behave entirely different through adding running time. Their searching mechanism largely determines their performance.

Although the results in Tables 4 to 6 have demonstrated the effectiveness of the proposed DIWO, due to the stochastic characteristic of the compared algorithms, it is essential to carry out a comprehensive statistical experiment to confirm whether the observed differences among the compared algorithms are statistically meaningful. In the statistical experiment, three types of statistical test are carried out, which include two parametric tests and a non-parametric test. Firstly, the parametric ANOVA test is used to check whether the ARPD obtained by all compared algorithms have statistically significant differences, where the heuristic type (algorithm) is considered as a factor. Therefore, we carry out three ANOVA tests in this experiment. All tests are run with a confidence level of 95% ($\alpha = 0.05$). The results of ANOVA are reported by using the form of ANOVA charts with 95% confidence intervals, which are shown in Fig. 9. It is remarkable that overlapping intervals of two algorithms indicate that there is no significant difference between their performance. It can be seen from Fig. 9 that there are no overlaps between DIWO and other compared algorithms for each termination criterion, which means the differences between DIWO and other algorithms are statistically significant at the 95% confidence level. Thus, we can conclude that DIWO significantly outperforms other compared methods for all termination criteria. Additionally, for $\rho = 60$ and $\rho = 90$, the confidence intervals of MA and P-EDA are almost completely overlapped, which reveals that they have similar performance on these termination criteria. TPA, HVNS, DE_PLS and SVNS_D also have overlapped confidence intervals when $\rho = 90$.

Secondly, the parametric Duncan's multiple range test is employed to grade all compared algorithms into several levels. Table 7 reports the results of grade, where we can see the proposed DIWO algorithm ranks the first level for each termination criterion, i.e., A, and there are no other algorithms in this level, which indicates the differences between DIWO with other compared algorithms are statistically significant and the proposed algorithm achieves the best performance among all compared algorithms. Meanwhile, MA and P-EDA are in the second level, which means they have similar performance. Moreover, DE-ABC is graded into the last level. As also seen from Table 7, the grade of TPA reduces with the increase of running time. For other algorithms, the changes of grade are very small.

Table 7
Results of Duncan's multiple range test ($\alpha = 0.05$).

Rank	$\rho = 30$	$\rho = 60$	$\rho = 90$
A	**{DIWO}**	**{DIWO}**	**{DIWO}**
B	{MA, P-EDA, TPA}	{MA, P-EDA}	{MA, P-EDA}
C	{SaDIWO}	{SaDIWO, HVNS, TPA}	{SaDIWO}
D	{SVNS_D, DE_PLS, HVNS}	{SVNS_D, DE_PLS}	{HVNS, TPA, DE_PLS}
E	{MFFO}	{MFFO}	{DE_PLS, TPA}
F	{hmgHS}	{RAIS, hmgHS}	{MFFO}
G	{HDDE}	{HDDE}	{RAIS, hmgHS}
H	{IG}	{IG}	{hmgHS, HDDE}
I	{RAIS}	{DE-ABC}	{IG}
J	{DE-ABC}	–	{DE-ABC}

To further statistically justify the performance of the competing algorithms, we also employ a powerful nonparametric statistical method, i.e., Holm's procedure (Holm, 1979) is employed. According to the Refs. Derrac et al. (2011) and García et al. (2009), Holm's procedure is more rigorous than the previous ones and very suitable for comparing evolutionary and swarm intelligence optimization algorithms. In Holm's procedure, we firstly assume that the proposed DIWO (i.e., the control algorithm) is equivalent to other 13 compared algorithms on each termination criterion. Thus, a total of 13 hypotheses are considered for each termination criterion, which are denoted by $H_1, H_2, \dots, H_{13}$. α is the probability of rejecting at least one H_i that is in fact true. p_i denotes the p-value of H_i. Holm's procedure orders the p-values of these hypotheses with an ascending order. Then if p-value (i.e., p_i) of H_i is lower than $\alpha/(k-i+1)$, the hypothesis H_i will be rejected, where i is the rank of H_i and k is the total number of hypotheses. The confidence level is set $\alpha = 0.05$. The results of Holm's procedure are given in Table 8. As seen in this table, when $\rho = 30$, the hypotheses DIWO=SaDIWO, DIWO=TPA, DIWO=P-EDA and DIWO=MA are accepted, which indicates DIWO is not significantly better than SaDIWO, TPA, P-EDA and MA on this termination criterion. For $\rho = 60$, only P-EDA and MA are not significantly inferior to DIWO. For $\rho = 90$, Holm's procedure rejects all of 13 pairwise comparisons, which highlights the fact that the proposed DIWO is a better algorithm than other 13 algorithms.

Although all explanations and details given in the original articles were strictly followed to implement these compared algorithms and closely reproduce the published results, to make the comparison results more convincing and reasonable, we also compare the results of DIWO algorithm with the reported results in the literature. The compared algorithms adopt the same performance evaluation metric (i.e., Eq. (18)) and test instances. These algorithms include HDDE (Wang et al., 2010a), hmgHS (Wang et al., 2011), IG (Ribas et al., 2011), RAIS (Lin and Ying, 2013), TPA (Wang et al., 2012), MA (Pan et al., 2013b), SVNS_S and SVNS_D (Ribas et al., 2013), HVNS (Moslehi and Khorasanian, 2014a), DE_PLS (Tasgetiren et al., 2015), and P-EDA (Shao et al., 2018a). Note that the results of HDDE and MA are obtained from (Tasgetiren et al., 2015), while the results of other algorithms are from their original literature. Additionally, in order to be in consistent with these compared algorithms, the termination criterion of the proposed DIWO algorithm is set $\rho = 100$. Each instance is independently run 10 times. The comparison results are reported in Table 9. As seen from this table, the overall average ARPD of the proposed DIWO algorithm is −0.85, which is superior to the corresponding average ARPD values of other compared algorithms. Probing further into the statistics in Table 9 shows that HDDE, RAIS, DE_PLS, P-EDA, and the proposed DIWO achieve the similar results for the small-scale instances (i.e., 20 jobs), while the results obtained by DE_PLS are slightly better than DIWO for middle scale instances (i.e., 50 jobs). Nevertheless, the DIWO algorithm outperforms other compared algorithms at a considerable margin for large scale instances (i.e., 200 and 500 jobs). Through comparing with the results in the literature, we can further confirm the proposed DIWO algorithm is an effective and efficient approach for BFSP with makespan criterion, especially for the larger scale BFSP. The superiority of the proposed algorithm is mainly attributed to the following aspects. (1)

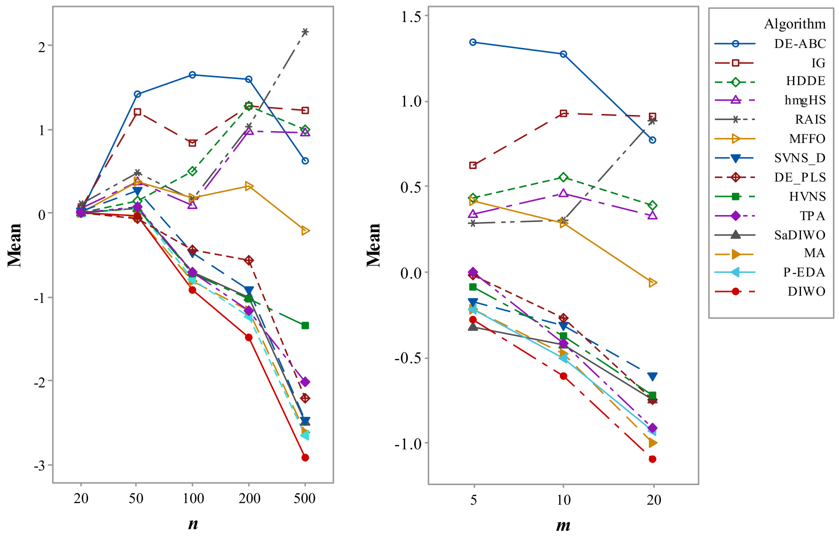

Fig. 7. Interaction plot between algorithm and n with m.

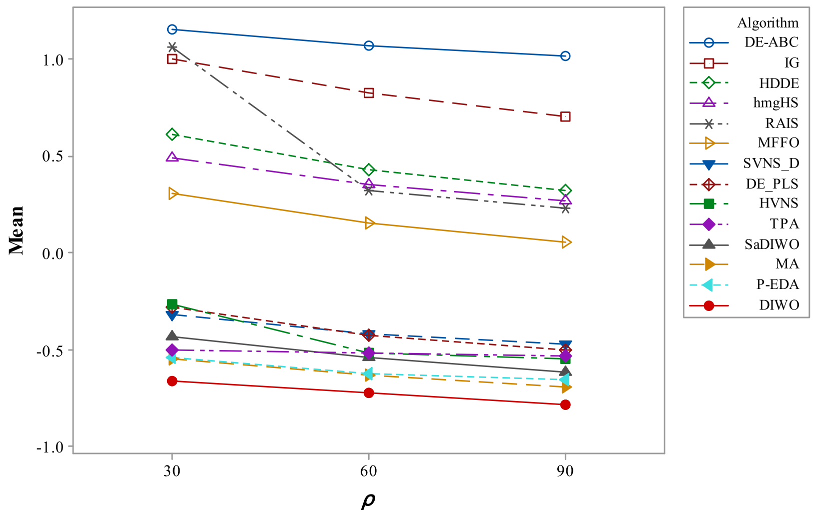

Fig. 8. Interaction plot between algorithm and ρ.

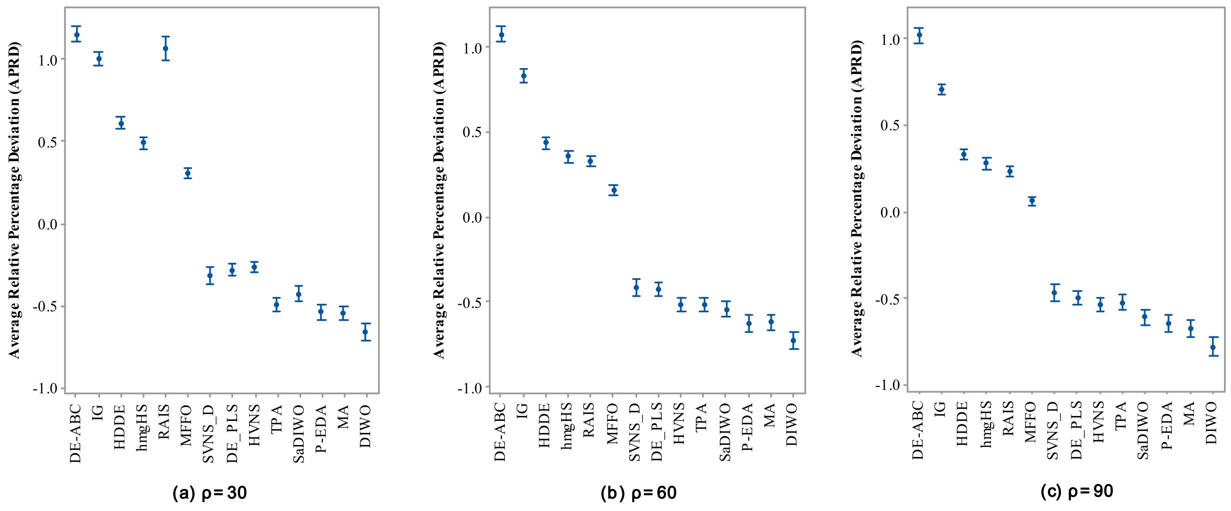

Fig. 9. Means plots and 95% confidence level intervals of all algorithms for different termination criteria.

Table 8
The results of Holm's procedure (α=0.05).

i	$\alpha/(k-i+1)$	ρ=30			ρ=60			ρ=90		
		H_i	p_i	R/A	H_i	p_i	R/A	H_i	p_i	R/A
1	0.0039	DIWO = DE-ABC	0.0000	R	DIWO = DE-ABC	0.0000	R	DIWO = DE-ABC	0.0000	R
2	0.0043	DIWO = RAIS	0.0000	R	DIWO = IG	0.0000	R	DIWO = IG	0.0000	R
3	0.0047	DIWO = IG	0.0000	R	DIWO = HDDE	0.0000	R	DIWO = HDDE	0.0000	R
4	0.0051	DIWO = HDDE	0.0000	R	DIWO = hmgHS	0.0000	R	DIWO = hmgHS	0.0000	R
5	0.0057	DIWO = hmgHS	0.0000	R	DIWO = RAIS	0.0000	R	DIWO = RAIS	0.0000	R
6	0.0064	DIWO = MFFO	0.0000	R	DIWO = MFFO	0.0000	R	DIWO = MFFO	0.0000	R
7	0.0073	DIWO = HVNS	0.0001	R	DIWO = SVNS_D	0.0006	R	DIWO = SVNS_D	0.0000	R
8	0.0085	DIWO = DE_PLS	0.0001	R	DIWO = DE_PLS	0.0009	R	DIWO = DE_PLS	0.0005	R
9	0.0102	DIWO = SVNS_D	0.0005	R	DIWO = TPA	0.0199	R	DIWO = TPA	0.0017	R
10	0.0127	DIWO = SaDIWO	0.0190	A	DIWO = HVNS	0.0209	R	DIWO = HVNS	0.0048	R
11	0.0170	DIWO = TPA	0.0980	A	DIWO = SaDIWO	0.0446	R	DIWO = SaDIWO	0.0078	R
12	0.0253	DIWO = P-EDA	0.2145	A	DIWO = P-EDA	0.0603	A	DIWO = P-EDA	0.0168	R
13	0.0500	DIWO = MA	0.2385	A	DIWO = MA	0.1161	A	DIWO = MA	0.0213	R

Note: R and A denote reject and accept hypothesis, respectively.

Table 9
The comparison results of the algorithms.

Instance	HDDE	hmgHS	IG	TPA	RAIS	SVNS_S	SVNS_D	HVNS	DE_PLS	MA	P-EDA	DIWO
20 × 5	**0.00**	0.09	0.39	0.40	**0.00**	0.12	0.16	0.01	**0.00**	**0.00**	**0.00**	**0.00**
20 × 10	**0.00**	0.03	0.48	0.21	**0.00**	0.15	0.13	0.01	**0.00**	0.01	**0.00**	**0.00**
20 × 20	**0.00**	0.03	0.31	0.03	**0.00**	0.08	0.07	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**
50 × 5	0.78	0.37	2.71	1.56	−0.19	0.69	0.77	0.17	**−0.28**	−0.17	−0.11	−0.10
50 × 10	0.53	0.50	3.24	1.07	−0.20	0.88	0.75	−0.06	**−0.38**	−0.18	−0.24	−0.18
50 × 20	0.35	0.29	2.88	0.76	0.00	0.85	0.70	−0.06	**−0.34**	−0.20	−0.12	−0.12
100 × 5	1.78	0.80	3.82	1.71	−0.82	−0.19	−0.14	−0.60	−0.81	−0.81	−0.96	**−0.98**
100 × 10	1.01	0.46	3.34	0.59	−0.88	0.11	0.14	−0.90	**−1.25**	−1.16	−1.21	−1.19
100 × 20	0.86	0.31	0.03	0.07	−0.71	0.37	0.44	−1.03	−1.14	−1.09	−1.02	**−1.17**
200 × 10	2.49	1.27	3.85	1.22	−0.33	0.04	0.04	−0.80	−0.93	−0.89	−1.41	**−1.44**
200 × 20	1.24	0.37	2.31	0.03	−0.63	−0.28	−0.27	−1.50	−0.95	−1.06	−1.82	**−1.87**
500 × 20	1.54	0.17	1.32	−0.05	−0.02	−1.75	−1.74	−2.00	−2.62	−2.65	−2.94	**−3.08**
Average	0.88	0.39	2.31	0.63	−0.32	0.09	0.086	−0.56	−0.73	−0.683	−0.82	**−0.85**

The adopted effective heuristic provides an excellent initial solution. (2) The proposed spatial dispersal operator effectively balances the global exploration and local exploitation. (3) The designed local intensification mechanism effectively exploits the areas around the new created seeds. (4) The adopted competitive exclusion maintains a good population diversity through eliminating the similar weeds in the colony.

The excellent performance of the proposed DIWO encourages us to find new best solutions of Taillard's benchmark instances for BFSP with makespan criterion. Most of algorithms for the considered problem have reported their best solutions in their original literature. Thus, we summary these reported results and compare with our proposed DIWO. Some new best solutions are discovered during experiment. Table 10 reports the comparison results, where the best solutions found by DIWO are highlighted in **bold**. In this table, the 'Source' column represents the methods adopted to obtain the best solution. In the 'Source' column, the numbers from '1' to '15' respectively represent HDDE (Wang et al., 2010a), DE-ABC (Han et al., 2015a), MFFO (Han et al., 2016b), IG (Ribas et al., 2011), TPA (Wang et al., 2012), TPA* (Moslehi and Khorasanian, 2014b), RAIS (Lin and Ying, 2013), SVNS (Ribas et al., 2013), MA (Pan et al., 2013a), MA* (Tasgetiren et al., 2015), HVNS (Moslehi and Khorasanian, 2014b), DE_PLS (Tasgetiren et al., 2015), DWWO (Shao et al., 2018b), P-EDA (Shao et al., 2018a), and the proposed DIWO. It should be noted that MA* and TPA* are the extensions of MA and TPA, respectively, which are run more CPU time than the original ones. It can be observed that all compared algorithms obtain the best values for the small-scale instances (i.e., 20 jobs). For the remaining 90 instances, the proposed DIWO provides 47 out of 90 new unique best solutions. Especially for large-scale instances, i.e., 200 × 10, 200 × 20, and 500 × 20, the proposed DIWO almost updates all of best solutions. The comparison results above further demonstrate that the proposed DIWO has good performance on solving the considered problem. Additionally, we also provide the sequences of the new best solutions on the online supplementary material, which can be a reference for future work along this line of research.

6. Conclusions and future research

In this paper, we propose a discrete invasive weed optimization for solving the blocking flow-shop scheduling problem with makespan criterion. In the proposed algorithm, an effective heuristic and the random method were combined to generate an initial plant population with high quality and diversity. A random-insertion-based spatial dispersal was developed by means of the normal distribution to guide the global exploration and local exploitation. A shuffle-based referenced local search was incorporated to strengthen local exploitation. An improved competitive exclusion was employed to guarantee the offspring population not only has good quality but also a certain level of diversity. The parameters setting was investigated by using a design of experiments approach. The effectiveness of the proposed spatial dispersal and local search was demonstrated by numerical comparisons. To verify the efficiency and effectiveness of the proposed algorithm, an extensive comparison was carried out. The comparison results showed that the proposed DIWO algorithm outperformed the recently published algorithms in terms of solution quality and search efficiency. Moreover, we have also updated 47 best solutions out of 120 Taillard's benchmark instances.

For the future research, the DIWO algorithm will be extended to solve the BFSP with other objectives, such as total flow time, total tardiness time and multi-objective BFSP. Additionally, we will try designing some adaptive ways to reduce the number of parameters for DIWO algorithm to further enhance the performance of it, and then apply it to solve more realistic scheduling problems.

Table 10
Best solutions for BFSP with makespan criterion.

Instance	Best	Source	DIWO	Instance	Best	Source	DIWO	Instance	Best	Source	DIWO
20 × 5				50 × 10				100 × 20			
Ta1	1374	1–15	1374	Ta41	3611	10	3614	Ta81	**7709**	**15**	**7709**
Ta2	1408	1–15	1408	Ta42	3470	14	3473	Ta82	7744	14	7769
Ta3	1280	1–15	1280	Ta43	**3465**	**15**	**3465**	Ta83	7723	14	7725
Ta4	1448	1–15	1448	Ta44	3650	8	3654	Ta84	7743	14	7755
Ta5	1341	1–15	1341	Ta45	3582	8	3619	Ta85	7730	13	7744
Ta6	1363	1–15	1363	Ta46	3571	14	3575	Ta86	7779	14	7785
Ta7	1381	1–15	1381	Ta47	3667	14	3669	Ta87	**7857**	**15**	**7857**
Ta8	1379	1–15	1379	Ta48	**3549**	**15**	**3549**	Ta88	7898	11	7902
Ta9	1373	1–15	1373	Ta49	3508	8	3518	Ta89	7818	14	7835
Ta10	1283	1–15	1283	Ta50	3608	12	3610	Ta90	**7842**	**15**	**7842**
20 × 10				50 × 20				200 × 10			
Ta11	1698	1–15	1698	Ta51	**4479**	**15**	**4479**	Ta91	13149	14	13166
Ta12	1833	1–15	1833	Ta52	4262	14, 15	4262	Ta92	**13085**	**15**	**13085**
Ta13	1659	1–15	1659	Ta53	4261	12, 15	4261	Ta93	**13183**	**15**	**13183**
Ta14	1535	1–15	1535	Ta54	4339	12	4342	Ta94	**13097**	**15**	**13097**
Ta15	1617	1–15	1617	Ta55	4249	12	4250	Ta95	**13100**	**15**	**13100**
Ta16	1590	1–15	1590	Ta56	4271	12, 15	4271	Ta96	**12883**	**15**	**12883**
Ta17	1622	1–15	1622	Ta57	4291	13, 14	4296	Ta97	**13408**	**15**	**13408**
Ta18	1731	1–15	1731	Ta58	4298	14, 15	4298	Ta98	**13266**	**15**	**13266**
Ta19	1747	1–15	1747	Ta59	4304	12	4305	Ta99	**13080**	**15**	**13080**
Ta20	1782	1–15	1782	Ta60	**4398**	**15**	**4398**	Ta100	**13193**	**15**	**13193**
20 × 20				100 × 5				200 × 20			
Ta21	2436	1–15	2436	Ta61	**6065**	**15**	**6065**	Ta101	14192	9	14516
Ta22	2234	1–15	2234	Ta62	**5934**	**15**	**5934**	Ta102	**14714**	**15**	**14714**
Ta23	2479	1–15	2479	Ta63	5851	8	5876	Ta103	**14831**	**15**	**14831**
Ta24	2348	1–15	2348	Ta64	5656	9	5684	Ta104	**14801**	**15**	**14801**
Ta25	2435	1–15	2435	Ta65	**5896**	**15**	**5896**	Ta105	**14609**	**15**	**14609**
Ta26	2383	1–15	2383	Ta66	**5755**	**15**	**5755**	Ta106	**14737**	**15**	**14737**
Ta27	2390	1–15	2390	Ta67	**5915**	**15**	**5915**	Ta107	**14770**	**15**	**14770**
Ta28	2328	1–15	2328	Ta68	5809	14	5825	Ta108	**14823**	**15**	**14823**
Ta29	2363	1–15	2363	Ta69	**6027**	**15**	**6027**	Ta109	**14691**	**15**	**14691**
Ta30	2323	1–15	2323	Ta70	6059	14, 15	6059	Ta110	**14711**	**15**	**14711**
50 × 5				100 × 10				500 × 20			
Ta31	2980	11	2983	Ta71	**6906**	**15**	**6906**	Ta111	**35380**	**15**	**35380**
Ta32	**3180**	**15**	**3180**	Ta72	**6656**	**15**	**6656**	Ta112	**35736**	**15**	**35736**
Ta33	2995	14	3000	Ta73	6797	14	6807	Ta113	**35406**	**15**	**35406**
Ta34	**3115**	**15**	**3115**	Ta74	**7035**	**15**	**7035**	Ta114	35030	13	35738
Ta35	3139	8	3145	Ta75	**6728**	**15**	**6728**	Ta115	**35417**	**15**	**35417**
Ta36	3158	13	3161	Ta76	6537	14	6549	Ta116	**35740**	**15**	**35740**
Ta37	3005	11	3008	Ta77	**6689**	**15**	**6689**	Ta117	**35299**	**15**	**35299**
Ta38	3042	11	3044	Ta78	6746	14,15	6746	Ta118	**35515**	**15**	**35515**
Ta39	2889	13,14	2891	Ta79	6928	10	6958	Ta119	**35268**	**15**	**35268**
Ta40	3097	14	3102	Ta80	6855	11	6881	Ta120	**35609**	**15**	**35609**

Acknowledgments

This research is supported by the National Natural Science Foundation of China [grant number U1433116]; the Fundamental Research Funds for the Central Universities [grant number NP2017208]; the Funding of Jiangsu Innovation Program for Graduate Education [grant number KYLX16_0382]; and the Postgraduate Research & Practice Innovation Program of Jiangsu Province [grant number KYCX17_0287].

Appendix A. Supplementary data

Supplementary material related to this article can be found online at https://doi.org/10.1016/j.engappai.2018.11.005.

References

Barisal, A.K., Prusty, R.C., 2015. Large scale economic dispatch of power systems using oppositional invasive weed optimization. Appl. Soft Comput. 29, 122–137.

Basak, A., Maity, D., Das, S., 2013. A differential invasive weed optimization algorithm for improved global numerical optimization. Appl. Math. Comput. 219, 6645–6668.

Črepinšek, M., Liu, S.H., Mernik, M., 2014. Replication and comparison of computational experiments in applied evolutionary computing: Common pitfalls and guidelines to avoid them. Appl. Soft Comput. 19, 161–170.

Dastranj, A., 2017. Optimization of a Printed UWB Antenna: Application of the invasive weed optimization algorithm in antenna design. IEEE Antennas Propag. Mag. 59, 48–57.

Davendra, D., BialicDavendra, M., 2013. Scheduling flow shops with blocking using a discrete self-organising migrating algorithm. Int. J. Prod. Res. 51, 2200–2218.

Davendra, D., Zelinka, I., Bialic-Davendra, M., Senkerik, R., Jasek, R., 2012. Clustered enhanced differential evolution for the blocking flow shop scheduling problem. CEJOR Cent. Eur. J. Oper. Res. 20, 679–717.

Deng, G.L., Zhang, S.N., Zhao, M., 2016. A discrete group search optimizer for blocking flow shop multi-objective scheduling. Adv. Mech. Eng. 8, 1–9.

Derrac, J., García, S., Molina, D., Herrera, F., 2011. A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms. Swarm Evol. Comput. 1, 3–18.

Ding, J.-Y., Song, S., Gupta, J.N.D., Wang, C., Zhang, R., Wu, C., 2016. New block properties for flowshop scheduling with blocking and their application in an iterated greedy algorithm. Int. J. Prod. Res. 54, 4759–4772.

Eddaly, M., Jarboui, B., Siarry, P., 2016. Combinatorial particle swarm optimization for solving blocking flowshop scheduling problem. J. Comput. Des. Eng. 3, 295–311.

Fernandez-Viagas, V., Leisten, R., Framinan, J.M., 2016. A computational evaluation of constructive and improvement heuristics for the blocking flow shop to minimise total flowtime. Expert Syst. Appl. 61, 290–301.

García, S., Molina, D., Lozano, M., Herrera, F., 2009. A study on the use of non-parametric tests for analyzing the evolutionary algorithms' behaviour: a case study on the CEC'2005 Special Session on Rea Parameter Optimization. J. Heuristics 15, 617–644.

Gong, H., Tang, L., Duin, C.W., 2010. A two-stage flow shop scheduling problem on a batching machine and a discrete machine with blocking and shared setup times. Comput. Oper. Res. 37, 960–969.

Grabowski, J., Pempera, J., 2000. Sequencing of jobs in some production system. European J. Oper. Res. 125, 535–550.

Graham, R.L., Lawler, E.L., Lenstra, J.K., Rinnooy Kan, A.H.G., 1979. Optimization and approximation in deterministic sequencing and scheduling: A survey. Ann. Discrete Math. 5, 287–326.

Hall, N.G., Sriskandarajah, C., 1996. A survey of machine scheduling problems with blocking and no-wait in process. Oper. Res. 44, 510–525.

Han, Y., Gong, D., Li, J., Zhang, Y., 2016a. Solving the blocking flow shop scheduling problem with makespan using a modified fruit fly optimisation algorithm. Int. J. Prod. Res. 54, 1–16.

Han, Y., Gong, D., Li, J., Zhang, Y., 2016b. Solving the blocking flow shop scheduling problem with makespan using a modified fruit fly optimisation algorithm. Int. J. Prod. Res. 54, 6782–6797.
Han, Y.Y., Gong, D., Sun, X., 2015a. A discrete artificial bee colony algorithm incorporating differential evolution for the flow-shop scheduling problem with blocking. Eng. Optim. 47, 1–20.
Han, Y.Y., Pan, Q.K., Li, J.Q., Sang, H.Y., 2011. An improved artificial bee colony algorithm for the blocking flowshop scheduling problem. Int. J. Adv. Manuf. Technol. 60, 1149–1159.
Han, Y.Y., Sun, D.G., Xiaoyan, 2015b. A discrete artificial bee colony algorithm incorporating differential evolution for the flow-shop scheduling problem with blocking. Eng. Optim. 47, 1–20.
Holm, S., 1979. A simple sequentially rejective multiple test procedure. Scand. J. Stat. 6, 65–70.
Jarboui, B., Eddaly, M., Siarry, P., Rebaï, A., 2009. An estimation of distribution algorithm for minimizing the makespan in blocking flowshop scheduling problems. In: Chakraborty, U.K. (Ed.), Computational Intelligence in Flow Shop and Job Shop Scheduling. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 151–167.
Karabulut, K., 2016. A hybrid iterated greedy algorithm for total tardiness minimization in permutation flowshops. Comput. Ind. Eng. 98, 300–307.
Koren, Y., Wang, W., Gu, X., 2017. Value creation through design for scalability of reconfigurable manufacturing systems. Int. J. Prod. Res. 55, 1227–1242.
Li, X., Wang, Q., Wu, C., 2009. Efficient composite heuristics for total flowtime minimization in permutation flow shops. Omega 37, 155–164.
Liang, J.J., Pan, Q.K., Chen, T., Wang, L., 2011. Solving the blocking flow shop scheduling problem by a dynamic multi-swarm particle swarm optimizer. Int. J. Adv. Manuf. Technol. 55, 755–762.
Lin, S.W., Ying, K.C., 2013. Minimizing makespan in a blocking flowshop using a revised artificial immune system algorithm. Omega 41, 383–389.
Mccormick, S.T., Pinedo, M.L., Shenker, S., Wolf, B., 1989. Sequencing in an assembly line with blocking to minimize cycle time. Oper. Res. 37, 925–935.
Mehrabian, A.R., Lucas, C., 2006. A novel numerical optimization algorithm inspired from weed colonization. Ecol. Inform. 1, 355–366.
Merchan, A.F., Maravelias, C.T., 2016. Preprocessing and tightening methods for time-indexed MIP chemical production scheduling models. Comput. Chem. Eng. 84, 516–535.
Montgomery, D.C., 2008. Design and Analysis of Experiments. John Wiley & Sons.
Moslehi, G., Khorasanian, D., 2014a. A hybrid variable neighborhood search algorithm for solving the limited-buffer permutation flow shop scheduling problem with the makespan criterion. Comput. Oper. Res. 52 (Part B), 260–268.
Moslehi, G., Khorasanian, D., 2014b. A hybrid variable neighborhood search algorithm for solving the limited-buffer permutation flow shop scheduling problem with the makespan criterion. Comput. Oper. Res. 52, 260–268.
Nagano, M.S., Komesu, A.S., Miyata, H.H., 2017. An evolutionary clustering search for the total tardiness blocking flow shop problem. J. Intell. Manuf. 1–15.
Niknamfar, A.H., Niaki, S.T.A., 2018. A binary-continuous invasive weed optimization algorithm for a vendor selection problem. Knowl.-Based Syst. 140, 158–172.
Nouri, N., Ladhari, T., 2017. Evolutionary multiobjective optimization for the multi-machine flow shop scheduling problem under blocking. Ann. Oper. Res. 1–18.
Pan, Q.-K., Ruiz, R., 2014. An effective iterated greedy algorithm for the mixed no-idle permutation flowshop scheduling problem. Omega 44, 41–50.
Pan, Q.K., Wang, L., 2012. Effective heuristics for the blocking flowshop scheduling problem with makespan minimization. Omega 40, 218–229.
Pan, Q.-K., Wang, L., Sang, H.-Y., Li, J.-Q., Liu, M., 2013a. A high performing memetic algorithm for the flowshop scheduling problem with blocking. IEEE Trans. Automat. Sci. Eng. 10, 741–756.
Pan, Q.K., Wang, L., Sang, H.Y., Li, J.Q., Liu, M., 2013b. A high performing memetic algorithm for the flowshop scheduling problem with blocking. IEEE Trans. Automat. Sci. Eng. 10, 741–756.
Pinedo, M.L., 2012. Scheduling: Theory, Algorithms, and Systems. Springer US.
Rama Prabha, D., Jayabarathi, T., 2016. Optimal placement and sizing of multiple distributed generating units in distribution networks by invasive weed optimization algorithm. Ain Shams Eng. J. 7, 683–694.
Rezaei Pouya, A., Solimanpur, M., Jahangoshai Rezaee, M., 2016. Solving multi-objective portfolio optimization problem using invasive weed optimization. Swarm Evol. Comput. 28, 42–57.
Riahi, V., Khorramizadeh, M., Hakim Newton, M.A., Sattar, A., 2017. Scatter search for mixed blocking flowshop scheduling. Expert Syst. Appl. 79, 20–32.
Ribas, I., Companys, R., Tort-Martorell, X., 2011. An iterated greedy algorithm for the flowshop scheduling problem with blocking. Omega 39, 293–301.
Ribas, I., Companys, R., Tort-Martorell, X., 2013. A competitive variable neighbourhood search algorithm for the blocking flow shop problem. Eur. J. Ind. Eng. 7, 729–754.
Ribas, I., Companys, R., Tort-Martorell, X., 2015. An efficient Discrete Artificial Bee Colony algorithm for the blocking flow shop problem with total flowtime minimization. Expert Syst. Appl. 42, 6155–6167.
Ronconi, D.P., 2004. A note on constructive heuristics for the flowshop problem with blocking. Int. J. Prod. Econ. 87, 39–48.
Roy, G.G., Das, S., Chakraborty, P., Suganthan, P.N., 2011. Design of non-uniform circular antenna arrays using a modified invasive weed optimization algorithm. IEEE Trans. Antennas Propag. 59, 110–118.
Sang, H.-Y., Duan, P.-Y., Li, J.-Q., 2018a. An effective invasive weed optimization algorithm for scheduling semiconductor final testing problem. Swarm Evol. Comput. 38, 42–53.
Sang, H.-Y., Pan, Q.-K., Duan, P.-Y., Li, J.-Q., 2018b. An effective discrete invasive weed optimization algorithm for lot-streaming flowshop scheduling problems. J. Intell. Manuf. 29, 1337–1349.
Saravanan, B., Vasudevan, E.R., Kothari, D.P., 2014. Unit commitment problem solution using invasive weed optimization algorithm. Int. J. Electr. Power Energy Syst. 55, 21–28.
Shao, W., Pi, D., 2016. A self-guided differential evolution with neighborhood search for permutation flow shop scheduling. Expert Syst. Appl. 51, 161–176.
Shao, Z., Pi, D., Shao, W., 2017. Self-adaptive discrete invasive weed optimization for the blocking flow-shop scheduling problem to minimize total tardiness. Comput. Ind. Eng. 111, 331–351.
Shao, Z., Pi, D., Shao, W., 2018a. Estimation of distribution algorithm with path relinking for the blocking flow-shop scheduling problem. Eng. Optim. 50, 894–916.
Shao, Z., Pi, D., Shao, W., 2018b. A novel discrete water wave optimization algorithm for blocking flow-shop scheduling problem with sequence-dependent setup times. Swarm Evol. Comput. 40, 1–23.
Taillard, E., 1993. Benchmarks for basic scheduling problems. European J. Oper. Res. 64, 278–285.
Tasgetiren, M.F., Kizilay, D., Pan, Q.-K., Suganthan, P.N., 2017. Iterated greedy algorithms for the blocking flowshop scheduling problem with makespan criterion. Comput. Oper. Res. 77, 111–126.
Tasgetiren, M.F., Pan, Q.K., Kizilay, D., Suer, G., 2015. A populated local search with differential evolution for blocking flowshop scheduling problem. In: 2015 IEEE Congress on Evolutionary Computation (CEC). IEEE, Sendai, Japan, pp. 2789–2796.
Wang, L., Pan, Q.-K., Suganthan, P.N., Wang, W.-H., Wang, Y.-M., 2010a. A novel hybrid discrete differential evolution algorithm for blocking flow shop scheduling problems. Comput. Oper. Res. 37, 509–520.
Wang, L., Pan, Q.K., Tasgetiren, M.F., 2010b. Minimizing the total flow time in a flow shop with blocking by using hybrid harmony search algorithms. Expert Syst. Appl. 37, 7929–7936.
Wang, L., Pan, Q.K., Tasgetiren, M.F., 2011. A hybrid harmony search algorithm for the blocking permutation flow shop scheduling problem. Comput. Ind. Eng. 61, 76–83.
Wang, C., Song, S., Gupta, J.N.D., Wu, C., 2012. A three-phase algorithm for flowshop scheduling with blocking to minimize makespan. Comput. Oper. Res. 39, 2880–2887.
Wang, X., Tang, L., 2012. A discrete particle swarm optimization algorithm with self-adaptive diversity control for the permutation flowshop problem with blocking. Appl. Soft Comput. 12, 652–662.
Zheng, Z.-x., Li, J.-q., 2018. Optimal chiller loading by improved invasive weed optimization algorithm for reducing energy consumption. Energy Build. 161, 80–88.
Zhou, Y., Luo, Q., Chen, H., He, A., Wu, J., 2015. A discrete invasive weed optimization algorithm for solving traveling salesman problem. Neurocomputing 151 (Part 3), 1227–1236.