Expert Systems With Applications 126 (2019) 41–53

Contents lists available at ScienceDirect

Expert Systems With Applications

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

A factorial based particle swarm optimization with a population adaptation mechanism for the no-wait flow shop scheduling problem with the makespan objective

Fuqing Zhao[a,*], Shuo Qin[a], Guoqiang Yang[a], Weimin Ma[b], Chuck Zhang[c], Houbin Song[a]

[a] School of Computer and Communication Technology, Lanzhou University of Technology, Lanzhou 730050, China
[b] School of Economics and Management, Tongji University, Shanghai 200092, China
[c] H. Milton Stewart School of Industrial & Systems Engineering, Georgia Institute of Technology, Atlanta, GA 30332, USA

ARTICLE INFO

Article history:
Received 19 October 2018
Revised 21 January 2019
Accepted 22 January 2019
Available online 14 February 2019

Keywords:
Particle swarm optimization
No-wait flow shop scheduling problem
Factorial representation
Runtime analysis
Variable neighborhood search
Makespan

ABSTRACT

The no-wait flow shop scheduling problem (NWFSP) performs an essential role in the manufacturing industry. In this paper, a factorial based particle swarm optimization with a population adaptation mechanism (FPAPSO) is implemented for solving the NWFSP with the makespan criterion. The nearest neighbor mechanism and NEH method are employed to generate a potential initial population. The factorial representation, which uniquely represents each number as a string of factorial digits, is designed to transfer the permutation domain to the integer domain. A variable neighbor search strategy based on the insert and swap neighborhood structure is introduced to perform a local search around the current best solution. A population adaptation (PA) mechanism is designed to control the diversity of the population and to avoid the particles being trapped into local optima. Furthermore, a runtime analysis of FPAPSO is performed with the level-based theorem. The computational results and comparisons with other state-of-the-art algorithms based on the Reeve's and Taillard's instances demonstrate the efficiency and performance of FPAPSO for solving the NWFSP.

1. Introduction

Shop scheduling plays an essential role in a manufacturing system as excellent scheduling planning improves the productivity of the company. The shop scheduling problem includes the single machine scheduling problem with a single processor, the single machine scheduling problem with parallel processors, the flow shop scheduling problem and the job shop scheduling problem (Zhao et al., 2017; Zhao et al., 2016, 2015). The flow shop scheduling problem (FSP) is a common model in the field of industrial production, and it is also a hot issue in academic research. The FSP is roughly divided into the following categories. The permutation FSP (PFSP) (Zhao et al., 2017), no-wait FSP (NWFSP) (Zhao et al., 2018), blocking FSP BFSP) (Ribas et al., 2017), no-idle FSP (NIFSP) (Shao et al., 2017a,b) and hybrid FSP (HFSP) (Yu et al., 2017).

In the NWFSP, n jobs are processed in the same order on each of m machines and there is no waiting time between two consecutive operations until the whole process is done. Therefore, the start time of a job on a machine has to be delayed to satisfy the no-wait constraints. The NWFSP has been proved to be a NP-hard problem when the number of machines is more than two with the objective of minimizing the makespan (Garey et al., 1976). It is difficult to solve the NWFSP by branch-and-bound or mixed integer programming methods with a reasonable time cost as the problem size increases. Therefore, it is necessary to try various efficient methods for solving the NWFSP.

Evolutionary algorithms (EA), which solve complex optimization problems without gradient information, have gained wide popularity. Recently, various EAs and the variants of classical EAs, such as particle swarm optimization (Zhao et al., 2015), harmony search (Zhao et al., 2017), biogeography-based optimization (Zhao et al., 2019), and the gravitational search algorithm (Zhao et al., 2018), have been proposed. Various heuristics are proposed to solve the NWFSP as they obtain high-quality solutions within a reasonable time limit. The heuristic algorithms are roughly classified into two main categories, constructive heuristics and meta-heuristics.

Various potential constructive heuristics have been developed over the last few decades to solve the NWFSP. The LC heuristic, which is based on the principle of job insertion, was designed by Laha et al. (2009) for minimizing makespan in the NWFSP. The experimental results demonstrated the LC heuristic outperforms four other compared algorithms in the litera-

* Corresponding author.
E-mail addresses: Fzhao2000@hotmail.com (F. Zhao), mawm@tongji.edu.cn (W. Ma), chuck.zhang@isye.gatech.edu (C. Zhang), 523712418@qq.com (H. Song).

https://doi.org/10.1016/j.eswa.2019.01.084
0957-4174/

ture. The average departure time (ADT) heuristic was proposed by Ye et al. (2016). The ADT outperforms the other compared algorithms on the large scale problems. An average idle time (AIT) heuristic, which achieved smaller deviations in the same computational complexity compared with the existing heuristics, was proposed by Ye et al. (2017) to minimize makespan in the NWFSP. In addition, several other constructive heuristics for the regular flow shop were also employed to solve the NWFSP, such as the NEH which was proposed by Nawaz et al. (1983).

Various noteworthy meta-heuristics have been developed for solving the NWFSP. A tabu-mechanism improved iterated greedy (TMIIG) algorithm was proposed by Ding et al. (2015). In the TMIIG, a tabu-based construction strategy and several neighborhood structures are employed to improve the quality of solutions. Recently, the block-shifting simulated annealing (BSA) algorithm, which combines a block-shifting mechanism and the framework of simulated annealing, was designed by Ding et al. (2015). The NWFSP model was formulated as the asymmetric traveling salesman problem (ATSP) by Lin et al. (2016). Two potential meta-heuristics were proposed to solve the NWFSP problem. An extended framework of a meta-heuristic based on the teaching-learning process was introduced by Shao et al. (2017). The computational results demonstrate that the proposed framework outperforms the other compared algorithms. The DWWO algorithm was designed by Zhao et al. (2018). In the DWWO, the operators of the original water wave algorithm were redefined to adapt to the NWFSP. A flower pollination algorithm was proposed by Qu et al. (2018) for solving the NWFSP.

The particle swarm optimization (PSO) algorithm, which mimics the swarm behaviors such as birds flocking and fish schooling, was proposed by Eberhart et al. (1995). Owing to the simple concept and high efficiency of PSO, the PSO algorithm has been successfully applied to various complex optimization problems. The SHSPSOS algorithm, which combines the harmony search algorithm and the particle swarm optimization, was proposed by Zhao et al. (2015) for solving the global numerical problem. The convergence performance of SHSPSOS was also analyzed with the Markov model. The particle swarm optimization algorithm with the cross operation (PSOCO) was proposed by Chen et al. (2018). The experimental results demonstrated that the PSOCO is a competitive optimizer regarding solution quality and efficiency. The PSO algorithm was also applied to solve the NWFSP. The discrete particle swarm optimization was presented by Pan et al. (2008) to solve the NWFSP with both the makespan and the total flow time criteria. The particle swarm optimization based on a memetic algorithm (MA) was proposed by Akhshabi et al. (2014) to solve the NWFSP. The experimental results demonstrate that the PSO-based MA is a robust algorithm.

Although the PSO algorithm and its variants were widely applied to solve the combinational optimization problems which include the NWFSP, little attention has been paid to employing the factorial representation to transfer the permutation domain to the integer domain. In this paper, a factorial particle swarm optimization algorithm with a population adaptation mechanism is proposed to solve the NWFSP with the minimization of the makespan objective. First, the factorial representation mechanism is introduced to transfer the permutation domain to the integer domain uniquely. Second, the NN+NEH method is employed to construct an initial population with desirable quality. Third, the VNS local search method is applied to exploit the promising area around the current best solution obtained by PSO. Finally, the population adaptation (PA) operator is designed to control the diversity of the population and to avoid the particles being trapped into local optima. The runtime of FPAPSO is also analyzed according to the level-based theorem. The experimental results based on the Reeve's and Taillard's instances demonstrate the effectiveness, efficiency and robustness of the proposed algorithm. The contributions of this paper are listed as follows.

- The NN+NEH method is introduced to initialize the potential population of PSO.
- The factorial representation is designed to transfer the permutation domain to the integer domain.
- The variable neighborhood search, which is based on the insert and swap neighborhood structure, is applied to search around the current best solutions in each generation.
- The population adaptation mechanism is designed to control the diversity of the population and avoid the particles being trapped into a local optimal.

This paper is organized as follows. Section 2 describes the model of the NWFSP. The proposed FPAPSO algorithm is introduced in detail in Section 3. In Section 4, the computational results on the well-known benchmark and comparisons are provided. Section 5 summarizes the conclusion and future research.

2. No-wait flow shop scheduling problem (NWFSP)

The NWFSP is described as follows. n jobs, which have the same processing routes, are processed through m machines, Each job j ($j = 1, 2, \ldots, n$) has a predefined processing time on each machine i ($i = 1, 2, \ldots, m$). A job is only processed at most by one machine and each machine executes one job at any moment. Each job is processed without waiting time between consecutive operations. The start of a job is delayed on the first machine to satisfy the no-wait constraint. In this paper, the goal is to find a feasible schedule π which has the minimum makespan for the n jobs in finite time.

The $\pi = [\pi(1), \pi(2), \ldots, \pi(n)]$ represents a schedule sequence. C_{max} denotes the makespan of π. $p(\pi(i), k)$ is the processing time of the ith job on the kth machine. The no-wait constraint ensures that the completion time distance between adjacent jobs is just related to the processing time of the two jobs. Thus, it is calculated between each pair of jobs. The completion time distance $D(i, j)$ from job i to job j is calculated as Eq. (1).

$$D(i,j) = \max_{k=1,\ldots m} \left\{ \sum_{h=k}^{m} (p(j,h) - p(i,h)) + p(i,k) \right\} \tag{1}$$

The makespan of the feasible schedule π is obtained as Eq. (2).

$$C_{max}(\pi) = \sum_{i=2}^{n} D(\pi(j-1), \pi(j)) + \sum_{k=1}^{m} p(\pi(1), k) \tag{2}$$

A virtual job whose processing time is set to zero is introduced to simplify the calculation process. The sequence π is replaced by $\pi' = [\pi(0), \pi(1), \pi(2), \ldots, \pi(n)]$. $D(\pi(0), \pi(1)) = \sum_{k=1}^{m} P(\pi(1), k)$. Therefore, the computational formula of makespan is simplified as Eq. (3).

$$\begin{aligned} C_{max}(\pi) &= \sum_{i=2}^{n} D(\pi(j-1), \pi(j)) + \sum_{k=1}^{m} p(\pi(1), k) \\ &= \sum_{i=2}^{n} D(\pi(j-1), \pi(j)) + D(\pi(0), \pi(1)) \\ &= \sum_{i=1}^{n} D(\pi(j-1), \pi(j)) \end{aligned} \tag{3}$$

The Π denotes the set of all possible permutations. The minimum makespan is described as follows.

$$C_{max}(\pi^*) = \min\{C_{max}(\pi) | \pi \in \Pi\} \tag{4}$$

3. The factorial representation based PSO with population adaptation (FPAPSO)

Symbols used mostly in this section are summarized as follows:

D_F	a list of possible digits of the factorial base in ascending order, $f=\{0, 1, ..., n-1\}$
PT	a factorial vector, $PT=\{(n-1)!, (n-2)!, ..., 1!, 0\}$
MRT	the maximum run time criteria of PSO
L	the population size of FPAPSO
x_g^i	the ith particle in the swarm at generation g
X_g	a set of L particles at generation g, $X_g=\{x_g^1, x_g^2, \ldots, x_g^L\}$
π_g^i	corresponding permutation of x_g^i
Π_g	corresponding permutations of X_g, $\Pi_g=\{\pi_g^1, \pi_g^2, \ldots, \pi_g^L\}$
V_g	velocity set of X_g

3.1. Original particle swarm optimization

Particle swarm optimization (PSO) (Eberhart et al., 1995) is a swarm intelligence algorithm, which simulates the behavior of the particle to find an optimal solution. In the PSO, each particle has two types of attribute. One is the position of the particle $X_i=(x_{i,1}, x_{i,2}, ..., x_{i,D})$. The other one is the velocity of particle $V_i=(v_{i,1}, v_{i,2}, ..., v_{i,D})$. The new velocity consists of the old velocity, the velocity towards the best position found by this particle and the velocity towards the global best position of the population. In the PSO, each particle maintains a memory to keep track of its previous best positions. The previous positions are distinguished as *pbest* and *gbest*. *pbest* is the previous best position found by particle X so far. *gbest* is the previous global best position found by the whole swarm so far. i is the number of iterations. The equation of velocity update is as follows.

$$V_{i+1} = wV_i + c_1 r_1 (pbest - X_i) + c_2 r_2 (gbest - X_i) \tag{5}$$

where w is the inertia factor and is calculated as follows.

$$w = w_{max} - \frac{w_{max} - w_{min}}{b} \times t \tag{6}$$

where t denotes the current number of generation, b denotes the total number of iterations. It is necessary to note that in the experimental phase of this paper, a maximum run time termination criteria is used to compare the algorithms. Therefore, w in the gth generation is calculated in a new way as follows.

$$w = w_{max} - \frac{w_{max} - w_{min}}{MRT} \times time_g \tag{7}$$

where $time_g$ denotes the run time from the beginning of the algorithm to the current generation, MRT is the maximum run time criteria. Eq. (7) produces the same effect as Eq. (6).

The c_1 and c_2 in Eq. (5) are positive constants, called the acceleration coefficients, r_1 and r_2 are two uniformly distributed random numbers in the interval [0, 1]. And the V_i belongs to $[-V_{max}, V_{max}]$.

The equation for position update is as follows.

$$X_{i+1} = X_i + V_{i+1} \tag{8}$$

The position of each particle is limited in [*LB, UB*]. Where *UB* is the upper bound, *LB* is the lower bound. The equation of judgment of the segment limit is as follows.

$$x_{i,j} \begin{cases} UB & x_{i,j} > UB \\ x_{i,j} & LB \le x_{i,j} \le UB \\ LB & x_{i,j} < LB \end{cases} \tag{9}$$

$$v_{i,j} \begin{cases} v_{max} & v_{i,j} > v_{max} \\ v_{i,j} & -v_{max} \le v_{i,j} \le v_{max} \\ -v_{max} & v_{i,j} < -v_{max} \end{cases} \tag{10}$$

In general, the limit of the maximum velocity v_{max} is selected empirically as the characteristics of the problem. In this paper, the maximum velocity v_{max} is calculated as follows.

$$v_{max} = (UB - LB)/L \tag{11}$$

where *UB* and *LB* are the upper bound and the lower bound respectively. L is the population size.

The full steps of the standard PSO are described as follows.

Step 1: Initialize the position and velocity of the population randomly.

Step 2: Update the velocity according to Eq. (5).

Step 3: Update the position according to Eq. (8).

Step 4: Judgment of the segment limit according to Eq. (9) and Eq. (10). Let the particle equal the *UB* if the particle is over the upper bound. Let the particle equal the LB if the particle is below the lower bound.

Step 5: Repeat Step 2 to Step 4 until the termination conditions are met.

In the following subsections, a factorial representation based particle swarm optimization with a population adaptation (FPAPSO) is proposed for solving the NWFSP. The three main operators, initial population, VNS local search and PA, are detailed as follows.

3.2. The factorial representation

The standard particle swarm optimization was designed to solve global numerical optimization problems. Therefore, the real-valued encoding schema was employed by the standard PSO. For the NWFSP, the real-number encoding schema is not suitable. In the discrete particle swarm optimization proposed by Pan et al. (2008), the permutation-based representation was employed to map the particle swarm optimization to the discrete search space. However, the framework of PSO has been modified as the inertia factor and acceleration coefficients were modified to probability instead of coefficients. The ranked-order-value (ROV) was employed by the hybrid particle swarm optimization (HPSO), which was introduced by Liu et al. (2007), for the NWFSP. Although the HPSO retains the framework of PSO, the search space was extended by ROV. In this paper, the factorial representation is employed to map the PSO for the NWFSP.

The factorial number system, called factorial system, is a factorial based mixed radix numeral system adapted to numbering permutations. Factorial representation was first presented by Laisant (1888). The idea of the factorial representation closely linked to that of the Lehmer code (Knuth, 1998). The factorial representation uniquely represents each number between 0 and $n!-1$ as a string of factorial digits. Each position i, $i \in [1, ..., n]$, is assigned a digit taking a value between 0 and i. The base of each position increases with i and so does its place value. The place value at position i is $(i-1)!$. The factorial $a_{(!)}$ is transformed into ith decimal form $a_{(10)}$ as follows.

$$a_{(10)} = \sum_{i=1}^{n} a_{(!)i} \times \ (i-1)! \tag{12}$$

where $a_{(!)i}$ denotes the ith element of $a_{(!)}$. The factorial represents a simple numbering system.

The factorial numbering system is unambiguous. Each number is represented in only one permutation as the sum of consecutive factorials multiplied by the index is always the next factorial minus one. There are n jobs, and the job sequence is $\pi=\{J_1 J_2,..., J_{j-1}, J_j,..., J_n\}$. There is a natural mapping between the integers 0, 1, ..., $n!-1$ (or equivalently the factorial numbers with n digits) and the permutations of n jobs in lexicographical order when the integers are expressed in factorial representation. The mapping has been

Table 1
The natural mapping between job permutations, factorial codes and decimal numbers when n = 3.

Job permutation	Factorial code	Decimal
{1, 2, 3}	{0, 0, 0}	0
{1, 3, 2}	{0, 1, 0}	1
{2, 1, 3}	{1, 0, 0}	2
{2, 3, 1}	{1, 1, 0}	3
{3, 1, 2}	{2, 0, 0}	4
{3, 2, 1}	{2, 1, 0}	5

Table 2
The encoding process of "[2, 3, 1] → [1, 1, 0] → 3".

Iteration (i)	i = 1	i = 2	i = 3
Permutation (J_i)	2	3	1
P	{1,2,3}	{1,3}	{1}
D_F	{0,1,2}	{0,1,2}	{0,1,2}
Factoradic (F)	1	1	0
Decimal (N)	$0+1\times 2!=\underline{2}$	$2+1\times 1!=\underline{3}$	$3+0\times 0!=\underline{3}$

termed the Lehmer code. For example, the complete mapping with $n = 3$ is shown in Table 1.

3.2.1. Factorial encoding schema

The encoding process is described as follows. First, consider a job permutation $\pi = \{J_1 J_2, \ldots, J_{j-1}, J_j, \ldots, J_n\}$, a list of possible digits of the factorial base in ascending order $D_F = \{0, 1, \ldots, n-1\}$, an ascending job list $P = \{1, 2, \ldots, n\}$ which contains all jobs, and an empty factorial sequence F. Second, the first job J_1 is chosen from the permutation π and the corresponding digit in P is found simultaneously. Suppose that the digit J_1 is the ith digit in P, the digit J_1 is removed from P and the corresponding ith digit f_i in D_F is put to the end of factorial list F. Therefore, the second job J_2 in π is carried out continuously and the above procedure is repeated until all jobs are accessed. Finally, the factorial list F is transformed to its decimal form N according to the Eq. (10). The pseudo code of the encode process is given in Algorithm 1. The computational complexity of the encode process is O(n).

The mapping of [2, 3, 1] → [1, 1, 0] → 3 based on Algorithm 1 is demonstrated in Table 2.

3.2.2. Factorial decode schema

The detailed decode process is devised in two phases. The factorial decimal form N is transformed to factorial in the first phase. The corresponding factorial is transformed into a job permutation π in the second phase.

The first phase is described as follows. Consider a factorial decimal form in N, a factorial vector $PT = \{(n-1)!, (n-2)!, \ldots, 1!, 0\}$ and an empty factorial list F. First, the ith digit PT_i is carried out from left to right in PT which is not greater than N, and put $i-1$ zero elements into F. Second, the quotient of N/PT_i is put into F and N is equal to the remainder of N/PT_i. The above two steps are repeated until N is equal to zero. Afterward, a factorial list of natural number N is obtained.

The second phase is described as follows. Consider a factorial list F obtained from the first phase, an empty job permutation π, a list of possible digits of the factorial base in ascending order $D_F = \{0, 1, \ldots, n-1\}$ and an ascending job list $P = \{1, 2, \ldots, n\}$ which contains all jobs. First, the first element F_1 in F is chosen, and this digit in D_F is found. Suppose that this digit D_{Fi} is the ith digit in D_F. The ith digit P_i in P is removed and put to the end of the job permutation list π. Second, the next element F_i in F is chosen and the above procedure is repeated until all the jobs are accessed. Finally, the corresponding job permutation is obtained. The pseudo code of the decoding process is given in Algorithm 2. The computational complexity of the encoding process is O(n).

Table 3
The decoding process of 3 → [1, 1, 0] → [2, 3, 1].

Iteration (i)	$i = 1$	$i = 2$	$i = 3$
Decimal (N)	$3 \div 2! = 1 \cdots \underline{1}$	$1 \div 1! = 1 \cdots \underline{0}$	$\underline{0}$
Factoradic (F)	1	1	0
D_F	{0,1,2}	{0,1,2}	{0,1,2}
P	{1,2,3}	{1,3}	{1}
Permutation (J_i)	2	3	1

The mapping of "3 → [1, 1, 0] → [2, 3, 1]" based on Algorithm 2 is demonstrated in Table 3.

3.3. Initial population

L initial solutions (or sequences) are generated by using the nearest neighbor (NN) (Fink et al., 2003) and the NEH (Nawaz et al., 1983) to accelerate the convergence speed of the early stages. L is the population size which is set by the author. The NN and the NEH are two popular heuristics. The NN heuristic appends an unscheduled job with a minimal delay time to the last job of the partial scheduled sequence at each step. The NEH heuristic consists of three steps. The three steps are described as follows.

Step 1: Sort the jobs according to the descending sums of their processing times and generate a sequence π.

Step 2: Take the first two jobs $J_1 J_2$ of π and evaluate the two possible sub-sequences. Then, the better sub-sequence is selected as the current sequence.

Step 3: Take job three and find the best sub-sequence by placing it in all possible positions of the sub-sequence that have been already scheduled. After that, repeat Step 3 with J_j ($j = 4, 5, \ldots, n$) until all jobs are sequenced.

The idea of using the NN heuristic and the NEH heuristic to construct an initial population was presented in the discrete particle swarm optimization (DPSO) algorithm. The NN+NEH are employed to obtain the L initial solutions in this paper. The detailed steps of NN+NEH are described as follows.

Step 1: Pick out L jobs $S = \{J_S^1, J_S^2, \ldots, J_S^L\}$ randomly from a universal set $P = \{J_P^1, J_P^2, \ldots, J_P^n\}$ which includes all of the jobs;

Step 2: The jth job J_j, $j = 1, 2, \ldots, L$, from S is taken as the first job in the jth candidate sequence π_j of the initial permutations on Π_0 which is denoted as $J_{\pi_j}^1$. Then, apply the NN heuristic to find a job $J_{\pi_j}^2$ with a minimal delay time to $J_{\pi_j}^1$;

Step 3: Apply the NEH heuristic with the other $n-2$ jobs from S_1 (i.e., exclude $J_{\pi_j}^1$ and $J_{\pi_j}^2$) to build a sub-sequence S_{NEH};

Step 4: The final permutation π_j is constructed by appending the S_{NEH} to the first two scheduled jobs $[J_{\pi_j}^1, J_{\pi_j}^2]$. Repeat the Step 2 and Step 3 until L initial permutations were obtained. The pseudo code of the initial population is given in Algorithm 3.

The pseudo code of the NN+NEH strategy is given in Algorithm 3. After L permutations are provided by the NN+NEH, Algorithm 1 is employed to code them to L integers as the initial population for PSO. Therefore, the PSO is able to start the search process in the integer domain.

Since the proposed FPAPSO algorithm uses particles to explore the search space, particle velocity is also needed to update the particle's position during iterations of the algorithm. The FPAPSO algorithm initially generates a random integer number as particle velocity to upgrade the particle's position. Note that velocities have to be in an appropriate interval so that the particle remains in the feasible space after being upgraded.

3.4. VNS local search

VNS is a recent meta-heuristic for combinatorial and global optimization problems (Mladenovi et al., 2008). The basic idea of the VNS is to allow a systematic change in neighborhood structures of the current best (incumbent) solution within a randomized local search. Its effectiveness was tested on several combinatorial problems.

The DPSO was proposed by Pan et al. (2008) for solving the NWFSP with both makespan and total flowtime criteria respectively. The VNS algorithm was embedded in the DPSO algorithm in the proposed DPSO to improve solution quality. The experimental results show that the VNS enhances solution quality substantially. Therefore, the VNS method is modified and embedded into the proposed FPAPSO algorithm in this paper. There are two structures of neighborhoods of the VNS, called the swap neighborhood and insert neighborhood, which are defined as follows:

(1) Swap two jobs between the ηth and kth dimensions, $\eta \neq k$ (*Swap*).
(2) Remove the job at the ηth dimension and insert it in the kth dimension $\eta \neq k$ (*Insert*).

The pseudo code of the VNS is given in Algorithm 4. An integer particle is decoded into a permutation by using Algorithm 2 to apply the VNS local search. In the same way, the local optimum permutation of VNS is encoded into an integer solution which is returned to PSO.

3.5. Population adaptation

Since each solution represented by factorial encoding schema has at most two neighbors, a local optimum appears when the adjacent three neighbors are arranged in a non-monotonically order. There are numerous local optima in the factorial representation. It means a naive PSO is easily trapped into a local optimum. Population adaptation (PA) is able to control the population's diversity when the population diversity is poor and enables the population to continue to evolve when it has been in stagnation. PA is designed to improve population diversity and avoid particles stacking into local optima in the PSO. The details of the PA are as follows.

Euclidean distance in the one-dimensional integer domain is employed to measure the population diversity in PA. Suppose that $X_g = \{x_1, x_2, \ldots, x_L\}$ is the population at the gth generation, where L denotes the population size. The sum of the Euclidean distances D_g between particles of X_g is calculated as follows.

$$D_g = \sum_{i=1}^{L-1} \sum_{j=i+1}^{L} \left| x_i - x_j \right| \tag{13}$$

When the population has converged at a local optimum, the population diversity is low. In this case, the change in D_g between two generations is not significant. If D_g remains unchanged over T consecutive generations, it represents that the PSO cannot generate any better trials to escape from the local optimum. Since the algorithm enters into a stable stagnation state will take more time to escape the local optimum when the population size becomes too large, T is set to equal to L instead of a constant. Then the algorithm needs to regenerate the population at this time. The new population $X'_g = \{x'_1, x'_2, \ldots, x'_L\}$ is generated as follows.

$$x'_i = N(\mu, \sigma)\ \ i = 1, 2, \ldots, L \tag{14}$$

where N denotes generating a normal distribution random number with a mean of μ and a variance of σ which were calculated as follows.

$$\mu = \frac{pbest - LB}{UB - LB} \tag{15}$$

$$\sigma = \left(1 - \frac{Time_g}{\mathrm{MRT}}\right) \cdot \max(\mu, 1 - \mu) \tag{16}$$

In Eq. (15), *pbest* is the best position among the individuals, *UB* and *LB* are the upper bound and lower bound of the search space respectively. From the encode scheme, the factorial code is the smallest value 0 when all the jobs are sorted in ascending order. The factorial code is the largest value $n! - 1$ when all the jobs are sorted in descending order. Therefore, the *LB* is set to 0 and the *UB* is set to $n! - 1$. In Eq. (16), $Time_g$ denotes the run time from the beginning of the algorithm to the current generation *g*. *MRT* is the maximum run time criteria.

Algorithm 5 illustrates the framework of the PA. Note that, the current best solution is not affected by the PA, it means the PA approach does not re-diversify the current best solution.

3.6. Proposed FPAPSO algorithm

The FPAPSO is a heuristic algorithm which is combined with PSO, PA and VNS to solve a factorial represented $F_m|nwt|C_{\max}$ problem. On the basis of the original PSO, the basic elements of FPAPSO are summarized as follows.

3.6.1. Particle (individual)

x_g^i denotes the ith particle in the swarm at generation g. Under the factorial representation, a particle is represented by a digit in the natural number domain. Note that a particle always corresponds to a job permutation.

3.6.1.1. Swarm (population). X_g is a set of L particles in generation g, $X_g = \{x_g^1, x_g^2, \ldots, x_g^L\}$.

3.6.1.2. Permutation. A particle x_g^i corresponds to a permutation π_g^i using encoding (see also Algorithm 1) and decoding (see also Algorithm 2) process. Similarly, a *swarm* X_g with L *particles* corresponds to a set Π_g, $\Pi_g = \{\pi_g^1, \pi_g^2, \ldots, \pi_g^L\}$ which contains L permutations π_g^i, $i = 1, 2, \ldots, L$.

3.6.1.3. Fitness value of particle. A particle has a fitness value which equals the makespan of the corresponding job replacement.

3.6.1.4. Search space. Assume there are n jobs, all the permutations are uniquely mapped to the natural numbers space: [0, 1, …, $n! - 1$]. The whole search space of FPAPSO is [0, 1, …, $n! - 1$].

3.6.1.5. Neighborhood. The neighborhood of FPAPSO is a natural number denoted (NND) neighborhood structure. Therefore, the two numbers around the current integer are the only two neighborhoods of the current particle.

Based on the above several important components of our proposed algorithm, the steps of FPAPSO algorithm are described below:

Step 1: *Initialize parameters*. Set the values of the control parameters: L (number of particles), *MRT* (maximum run time criteria), c_1 and c_2 (velocity constants), w_{min} and w_{max} (parameters to affect inertia weight), $c = 0$ (population distance unchanged times), $g = 1$ (the current number of iterations).

Step 2: *Initialize population*. Generate a velocity set L initial permutations using the NN+NEH detailed in Algorithm 3. Evaluate the permutations to get the *pbest, gbest* and D_0. Then the L permutations must be encoded as L integers using the encoding process detailed in Algorithm 1. The L integers form the initial population. Finally, randomly generate a feasible initial velocity set.

Step 3: *Move the particles*. Update the population by Eq. (8). Map the new population to their corresponding permutation and

evaluate the makespan of each new permutation. Update the variables *pbest* and *gbest*.

Step 4: *VNS local search*. Get a local optimal permutation by using the VNS local search algorithm (detailed in Algorithm 4) for the permutation of the *pbest* solution. Then replace *pbest* particles with the corresponding integer of the local optimal permutation.

Step 5: *Population adaptation*. Apply the PA algorithm (detailed in Algorithm 5) for the current population to get the adapted population. Then replace the current population with the adapted population corresponding integers.

Step 6: *Check termination condition*. If the termination condition is met, stop. Return the value of variable *gbest* and the corresponding permutation as a final solution. Otherwise, proceed to Step 7.

Step 7: *Update the particle velocity*. Update the particle velocity set using Eq. (5). $g=g+1$, Go to Step 3.

The pseudo code of the proposed algorithm is presented in Algorithm 6.

3.7. Runtime analysis of FPAPSO

In EAs, it is a fundamental problem to analyze the impact of parameter settings and the characteristics of the fitness landscape on time complexity. The level-based theorem is a technique tailored to population-based algorithms. The level-based theorem provides the upper bound of expected time that the algorithm discovers an element in the last level A_m when the following conditions are satisfied. First, the probability of creating an individual at level $j+1$ or higher is at least z_j when some individuals of the population have reached the level j or a higher level. Second, the number of individuals at level $j+1$ tends to increase. Finally, the population is sufficiently large. In this subsection, the upper bound on the runtime of the FPAPSO is analyzed with the level-based theorem.

Level-based Theorem (Corus et al., 2014). Given a partition $(A_1,\cdots, A_m)$ of χ, define T: $=\min\{t\lambda||P_t\cap A_m|>0\}$, where for all $t\in\mathbb{N}$, $P_t\in\chi^\lambda$ is the population in generation t. If there exist $z_1,\ldots, z_{m-1}$, $\delta\in(0,1]$, and $\gamma_0\in(0,1)$ such that for any population $P\in\chi^\lambda$,

(G1) for each level $j\in[m-1]$, if $|P\cap A_{\geq j}|\geq\gamma_0\lambda$, then

$$\Pr\left(y\in A_{\geq j+1}\right)\geq z_j,$$

(G2) for each level $j\in[m-2]$, and all $\gamma\in(0,\gamma_0]$, if $|P\cap A_{\geq j}|\geq\gamma_0\lambda$ and $|P\cap A_{\geq j+1}|\geq\gamma\lambda$, then

$$\Pr\left(y\in A_{\geq j+1}\right)\geq(1+\delta)\gamma,$$

(G3) the population size $\lambda\in$ satisfies

$$\lambda\geq\left(\frac{4}{\gamma_0\delta^2}\right)\ln\left(\frac{128m}{z_*\delta^2}\right)$$

where $z_*:=\min_{j\in[m-1]}\{z_j\}$, then

$$E[T]\leq\left(\frac{8}{\delta^2}\right)\sum_{j=1}^{m-1}\left(\lambda\ln\left(\frac{6\delta\lambda}{4+z_j\delta\lambda}\right)+\frac{1}{z_j}\right).$$

Definition. . Given a partition $(A_1,\cdots, A_m)$, if $(\ell_1,\ldots,\ell_\lambda)$ is employed to denote the sorted level of search points in P, the probability of selection is

$$\zeta(\gamma,P):=\sum_{i=1}^{\lambda}p(i|P)\cdot\left[P(i)\in A_{\geq\ell_{\gamma\lambda}}\right]$$

Corollary. . Given a partition $(A_1,\cdots, A_m)$ of χ, define T: $=\min\{t\lambda||P_t\cap A_m|>0\}$, where for all $t\in\mathbb{N}$, $P_t\in\chi^\lambda$ is the population in generation t. There exist $s,\ldots,s_{m-1}$, s_*, p_0, ε, $\delta\in(0,1]$, and a constant $\gamma_0\in(0,1]$ such that

(C1) for each level $j\in[m-1]$

$$P_{FPAPSO}(y\in A_{\geq j+1}|x\in A_j)\geq s_j$$

(C2) for each level $j\in[m-1]$

$$P_{FPAPSO}(x\in A_{\geq j}|x\in A_j)\geq p_0$$

(C3) for any population $P\in(\chi\backslash A_m)^\lambda$ and $\gamma\in(0,\gamma_0]$

$$\zeta(\gamma,P)\geq\frac{(1+\delta)\gamma}{p_0}$$

(C4) the population size satisfied

$$\lambda\geq\left(\frac{4}{\gamma_0\delta^2}\right)\ln\left(\frac{128m}{\gamma_0\delta^2 s_*}\right)$$

where $s_*:=\min_{j\in[m-1]}\{s_j\}$, then,

$$E[T]\leq\left(\frac{8}{\delta^2}\right)\sum_{j=1}^{m-1}\left(\lambda\ln\left(\frac{6\delta\lambda}{4+\gamma_0 s_j\delta\lambda}\right)+\frac{1}{\gamma_0 s_j}\right).$$

Proof. . According to the guidelines provided by Corus, et al. (2014), the level-based theorem is applied.

Step 1: The partition $A_j:=\{j\}$ for all $j\in[m]$ is employed, where A_m is the goal state.

Step 2: Assume that $|P\cap A_{\geq j}|>\gamma_0\lambda$ and $|P\cap A_{\geq j+1}|\geq\gamma\lambda>0$.

It suffices to pick the individual $x\in|P\cap A_k|$ for any $k\geq j+1$ from the personal best library and mutate it by FPAPSO to an individual in $A_{\geq k}$. As (C2) and (C3), the probability of such an event is at least

$$\zeta(\gamma,P)p_0\geq(1+\delta)\gamma$$

Condition (G2) is satisfied with the same γ_0 and δ as in (C3).

Step 3: Assume that $|P\cap A_j|\geq\gamma_0\lambda$. The individual x_k for any $k\geq j+1$ is chosen with probability $\zeta(\gamma_0,P)$.

If $x\in A_j$, then the FPAPSO will by (M1) upgrade x to $A_{\geq j+1}$ with s_j.

If $x\in A_{\geq j+1}$, the FPAPSO leaves the individual x in $A_{\geq j+1}$ with probability with p_0.

Then the probability of producing the individual in $A_{\geq j+1}$ is at least

$$\zeta(\gamma_0,P)\min\left\{s_j,p_0\right\}\geq\zeta(\gamma_0,P)s_jp_0>\gamma_0 s_j$$

Condition (G1) is satisfied with $z_j=\gamma_0 s_j$ and $z_*=\gamma_0 s_*$

Step 4: Given that $z_*=\gamma_0 s_*$, (C4) implied (G3).

Step 5: Condition (G1-3) are satisfied, the level-based theorem gives

$$E[T]\leq\left(\frac{8}{\delta^2}\right)\sum_{j=1}^{m-1}\left(\lambda\ln\left(\frac{6\delta\lambda}{4+\gamma_0 s_j\delta\lambda}\right)+\frac{1}{\gamma_0 s_j}\right)$$

The expected time to reach the last level A_{m+1} is less than

$$\left(\frac{8}{\delta^2}\right)\sum_{j=1}^{m-1}\left(\lambda\ln\left(\frac{6\delta\lambda}{4+\gamma_0 s_j\delta\lambda}\right)+\frac{1}{\gamma_0 s_j}\right)$$

However, a general formula, which is employed to calculate the upper bounds of expected time, is provided instead of a determined upper bound. The upper bounds of various instances are different since the different parameter settings and the different characteristics of the fitness landscape. Besides, the motivation of this paper is to provide an algorithm which finds better solutions within a limited time. The upper bound of each instance is not suitable to set the runtime stop criterion in the experiments.

4. The experimental results and analysis

The FPAPSO algorithm for the NWFSP was coded using Java. The simulation experiments were carried out on a personal computer (PC) with Intel (R) Core (TM) i7-6700 CPU 3.4 GHz and 8.00GB memory with a Windows Server 2012 Operating System.

In this section, the FPAPSO algorithm is compared with the state-of-the-art algorithms. 141 instances are employed to test the considered algorithms: (i) 21 instances provided by Reeves (1995): Rec01 to Rec41 that comprises seven sub-sets of different sizes which range from 20 jobs and 5 machines to 75 jobs and 20 machines. (ii) 120 instances provided by Taillard (1993): Ta001-Ta120 consists of 12 subsets of different sizes which range from 20 jobs and 5 machines to 500 jobs and 20 machines. The relative deviation (RD) between the solutions from the algorithms and the best-known solutions which was provided by Lin et al. (2016) were collected to compare the performance of FPAPSO with the other algorithms visually. The average relative deviation (ARD) and the best relative deviation (BRD) are employed to measure the quality of the experimental results. The value of ARD and BRD are calculated as follows.

$$ARD = \frac{1}{R}\sum_{r=1}^{R}\frac{C_r - C_R^*}{C_R^*} \times 100 \tag{17}$$

$$BRD = \frac{C_{best} - C_R^*}{C_R^*} \times 100 \tag{18}$$

where C_R^* denotes the best solutions, which are provided by Lin et al. (2016). In this paper, each testing case is executed 30 independent times for comparison. Therefore, the number of independent runs R is set to 30. C_r is the solution of the rth experiment in 30 independent runs. C_{best} is the best solution found over R runs, namely, C_{best} is the minimum value of C_r.

This section includes two subsections. The parameters setting of FPAPSO is discussed in the first subsection. In the second subsection, the performance of FPAPSO, HMM-FPA, DWWO, TMIIG, DPSO and IIGA are tested on the 21 Reeve instances and the 120 Taillard's instances.

4.1. Parameter setting

Five key control parameters should be tuned for the proposed algorithm to initiate the search. Values of five parameters affect the performance of the algorithm. A sensitivity analysis of the five parameters has been performed to determine the effect of the different parameter combinations on the performance of the algorithm. Based on the parameter sensitivity analysis approach, four problems of different sizes (20 × 10, 50 × 20, 100 × 20 and 200 × 20) are chosen from Taillard's instances. Then, the problems are solved with different combinations of parameter values. The multi-factor analysis of variance (ANOVA) method is introduced to investigate the experimental results. Based on the ANOVA results, the following experimentally derived values are proposed for the parameters: $L = n$, $c_1 = 2$, $c_2 = 2$, $w_{\max} = 1$, $w_{\min} = 0.5$.

4.2. Results and analysis

In the following experiment, the proposed FPAPSO is compared with state-of-the-art algorithms for solving the NWFSP with the minimization of the makespan criterion. The compared algorithms are HMM-FPA Qu et al., 2018), DWWO (Zhao et al., 2018), TMIIG (Ding et al., 2015a,b), $DPSO_{VND}$ (Pan et al., 2008a,b) and IIGA (Pan et al., 2008a,b). Besides, the PAPSO is also designed to perform the effectiveness of factorial representation. The only difference between the FPAPSO and PAPSO is that the PAPSO uses a permutation representation instead of factorial representation. For these algorithms, two standard benchmark sets of different scales are applied to test the performance of the above algorithms. (i) 21 instances designed by Reeves (1995): Rec01 to Rec41 that are divided into seven subsets of different size problems, ranging from 20 × 5 to 75 × 20. ((3) 120 instances designed by Taillard (1993): Ta001 to Ta110 that range from 20 × 5 to 200 × 20. These instances are available from the OR library website. Each instance is independently run 30 times for every algorithm for comparison.

The algorithms chosen for comparison except for the HMM-FPA have been carefully re-implemented in Java by all the details given in the original papers which are run on the same platform and with the same maximum run time (MRT) termination condition used by the FPAPSO. The results of HMM-FPA are carried out from the original paper. In various papers, the MRT value is only related to the number of jobs. However, the amount of computation increases as the number of machines increases. In this paper, the MRT condition of each compared algorithm is set to $n/2 \times m \times 30$ milliseconds (ms). In this experiment, all the algorithms converged slowly or stagnated when the run time reaches MRT.

For the Reeves instances, the computational results obtained by the six compared algorithms are listed in Table 4. The best results of each instance are given in boldface. From Table 4, most of the BRD values obtained by the FPAPSO are better than the other compared algorithms or at least equal only except in the Rec41 instance. On the Rec41 instance, the BRD value 0.07 yielded by the FPAPSO algorithm is greater than the corresponding value 0.00 obtained by HMM-FPA. Besides, the ARD values obtained by the FPAPSO are better than the other compared algorithms or at least equal. The ARD of the 21 Reeves instances are shown in Fig. 1. From Fig. 1, the performance of FPAPSO outperforms the other state-of-the-art algorithms on Reeve's instances.

The Taillard's instances are carried out to further demonstrate the performance of FPAPSO for solving the large-scale problems. The computational results of six compared algorithms are summarized in Table 5. The best results of each measure are given in boldface. From Table 5, The average BRD value 0.42 obtained by the FPAPSO is better than the corresponding value 0.52, 0.76, 0.56, 0.97, 0.72 obtained by HMM-FPA, DWWO, TMIIG, DPSOVND and IIGA, respectively. The ARD value 0.58 obtained by FPAPSO is also better than the 0.74, 0.95, 0.67, 1.15, 0.95 obtained by other state-of-the-art algorithms. For the large-scale instances, FPAPSO is superior to the other algorithms under the same run time. The average convergence curves of the algorithms on TA60, TA80, TA100 and TA120 are shown in Fig. 2 to demonstrate the convergence performance of FPAPSO clearly. From the Fig. 2, the FPAPSO has faster convergence speed than the compared algorithms on the different size Taillard's instances. Fig. 3 and Fig. 4 are the Gantt charts for the optimal results obtained by FPAPSO for TA01 and TA11. The ARD of 12 different scales of the Taillard's instances is shown in Fig. 5. From the Fig. 5, the performance of FPAPSO outperforms the other state-of-the-art algorithms on Taillard's instances.

Rigorous statistical studies based on the ARD values for the 120 Taillard's instances are carried out to verify the effectiveness of the FPAPSO for solving the NWFSP. The Friedman's test (García et al., 2009) is performed to rank the algorithms. The Friedman's test is implemented based on the SPSS software. The average ranking of the above six algorithms obtained by Friedman's test is summarized in Table 6. As shown in Table 6, there is a statistically significant difference in the optimization results depending on which type of algorithm is chosen, $\chi^2(2) = 351.806$, $p = 6.3191E-73$ with $\alpha = 0.05$ and $\alpha = 0.01$. FPAPSO has the best ranking among the six algorithms. The additional Bonferroni-Dunn's method is applied as a post hoc procedure to evaluate the significance level of all the algorithms.

$$CD = q_\alpha\sqrt{\frac{k(k+1)}{6N}}. \tag{19}$$

In Eq. (19), parameters k and N are the number of algorithms and number of instances, respectively. They are $k = 6$ and $N = 120$ in the experimental evaluations. When $\alpha = 0.05$, q_α is 2.576 and when $\alpha = 0.01$, q_α is 3.091 from Table B.16 (two-tailed $\alpha(2)$) of

Table 4
Comparison of results based on Reeve's benchmark set.

Instance		FPAPSO		PAPSO		HMM-FPA		DWWO		TMIIG		$DPSO_{VND}$		IIGA	
		BRD	*ARD*	*BRD*	*ARD*	*BRD*	*ARD*	*BRD*	*ARD*	*BRD*	*ARD*	*BRD*	*ARD*	*BRD*	*ARD*
Rec01	1526	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**
Rec03	1361	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**
Rec05	1511	**0.00**	**0.00**	**0.00**	0.05	**0.00**	0.04	**0.00**	0.01	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**
Rec07	2042	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	0.01	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**
Rec09	2042	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**
Rec11	1881	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**
Rec13	2545	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	0.03	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**
Rec15	2529	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**
Rec17	2587	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**
Rec19	2850	**0.00**	**0.00**	**0.00**	0.16	**0.00**	**0.00**	**0.00**	0.40	**0.00**	**0.00**	**0.00**	0.11	**0.00**	0.03
Rec21	2821	**0.00**	0.11	**0.00**	0.16	**0.00**	0.15	**0.00**	0.16	**0.00**	**0.03**	**0.00**	0.11	**0.00**	0.19
Rec23	2700	**0.00**	**0.00**	**0.00**	0.14	**0.00**	0.01	**0.00**	0.13	**0.00**	**0.00**	**0.00**	0.07	**0.00**	0.05
Rec25	3593	**0.00**	**0.00**	**0.00**	0.09	**0.00**	0.09	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	0.17	**0.00**	**0.00**
Rec27	3431	**0.00**	**0.00**	**0.00**	0.21	**0.00**	0.07	**0.00**	0.02	0.32	0.32	**0.00**	0.26	**0.00**	0.12
Rec29	3291	**0.00**	**0.00**	**0.00**	0.07	**0.00**	0.13	**0.00**	0.24	**0.00**	0.14	**0.00**	0.06	**0.00**	**0.00**
Rec31	4307	**0.00**	**0.11**	0.39	0.77	0.13	0.20	0.23	0.41	0.09	0.29	0.12	0.39	0.51	0.78
Rec33	4424	**0.00**	**0.23**	0.09	1.00	**0.00**	0.54	0.27	0.62	**0.00**	0.40	0.25	0.72	0.79	1.24
Rec35	4397	**0.00**	**0.12**	0.39	1.22	**0.00**	0.83	0.14	0.39	**0.00**	0.21	**0.00**	0.39	0.41	0.78
Rec37	8008	**0.00**	**0.36**	0.67	1.07	0.25	0.67	0.24	0.48	0.31	0.55	0.27	0.69	1.14	1.43
Rec39	8419	**0.09**	**0.33**	0.44	0.96	0.20	0.65	0.40	0.67	0.17	0.48	0.19	0.70	0.91	1.34
Rec41	8437	0.07	**0.41**	0.65	0.98	**0.00**	0.69	0.17	0.49	0.07	0.48	0.36	0.71	1.11	1.24

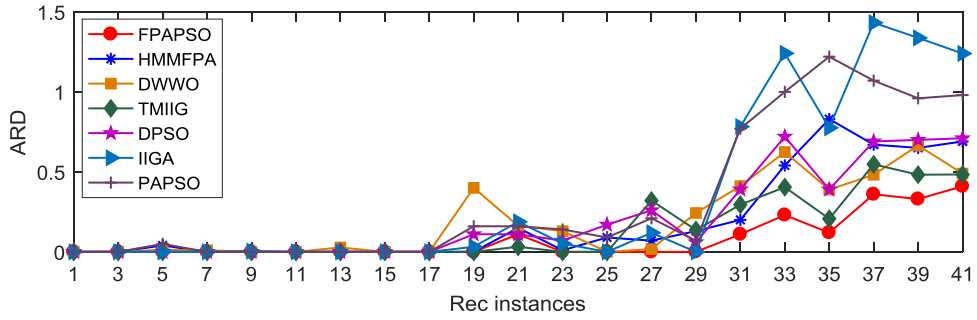

Fig. 1. The ARD of FPAPSO, FPAPSO, HMMFPA DWWO, TMIIG, DPSO, IIGA for Rec instances.

Table 5
Comparison of results based on Taillard's benchmark set.

n × m	FPAPSO		PAPSO		HMM-FPA		DWWO		TMIIG		$DPSO_{VND}$		IIGA	
	BRD	*ARD*	*BRD*	*ARD*	*BRD*	*ARD*	*BRD*	*ARD*	*BRD*	*ARD*	*BRD*	*ARD*	*BRD*	*ARD*
20 × 5	**0.00**	**0.00**	**0.00**	0.01	**0.00**	**0.00**	**0.00**	0.01	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**
20 × 10	**0.00**	**0.00**	**0.00**	0.01	**0.00**	**0.00**	**0.00**	0.01	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	0.00
20 × 20	**0.00**	**0.00**	**0.00**	0.01	**0.00**	**0.00**	**0.00**	0.01	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	0.01
50 × 5	0.16	**0.37**	0.62	0.95	0.22	0.37	0.26	0.50	0.33	0.47	0.58	0.79	**0.13**	**0.37**
50 × 10	**0.01**	0.20	0.27	0.68	0.04	**0.18**	0.17	0.39	0.15	0.27	0.47	0.69	0.08	0.22
50 × 20	**0.02**	**0.12**	0.26	0.56	0.05	0.37	0.07	0.31	0.17	0.27	0.34	0.64	0.05	0.22
100 × 5	**0.54**	0.85	1.93	2.36	0.62	1.13	2.29	2.65	0.68	**0.79**	1.26	1.53	0.92	1.26
100 × 10	**0.40**	**0.65**	1.26	1.77	0.47	0.97	0.51	0.74	0.50	0.65	0.95	1.25	0.58	0.87
100 × 20	**0.32**	0.63	1.02	1.35	0.41	0.69	0.35	**0.56**	0.49	0.72	0.95	1.21	0.50	0.81
200 × 10	1.24	1.47	3.06	3.51	1.26	1.49	1.98	2.34	**1.16**	**1.34**	2.03	2.26	1.86	2.18
200 × 20	**0.83**	**0.98**	2.33	2.72	1.04	1.36	0.97	1.22	1.07	1.22	1.79	2.07	1.34	1.60
500 × 20	**1.53**	**1.71**	4.38	4.71	2.10	2.31	2.45	2.69	2.19	2.32	3.26	3.41	3.13	3.46
Avg	**0.42**	**0.58**	1.26	1.55	0.52	0.74	0.76	0.95	0.56	0.67	0.97	1.15	0.72	0.92

Table 6
Ranking of the compared algorithm obtained through Friedman's test.

Algorithm	Mean Rank	Chi-Square	p-value
FPAPSO	2.45		
PAPSO	6.26		
HMMFPA	3.17	351.806	6.3191E-73
DWWO	4.06		
TMIIG	2.93		
DPSO	5.23		
IIGA	3.92		
Crit. Diff.	0.7360		
Crit. Diff.	0.8768		

(Zar, 1999). Fig. 6 sketches the results of Bonferroni-Dunn's test considering the FPAPSO as the control algorithm. There is a significant difference between the FPAPSO and the other compared algorithms. In summary, the above comparison clearly demonstrates that the FPAPSO is significantly better than the other compared algorithms.

The multiple-problem Wilcoxon's test (García et al., 2009) is performed to further check the behaviors of the above six algorithms. The statistical analysis results are summarized in Table 7. The FPAPSO is considered as the control algorithm. From Table 7, the FPAPSO provides higher $R+$ values than $R-$ values in all the

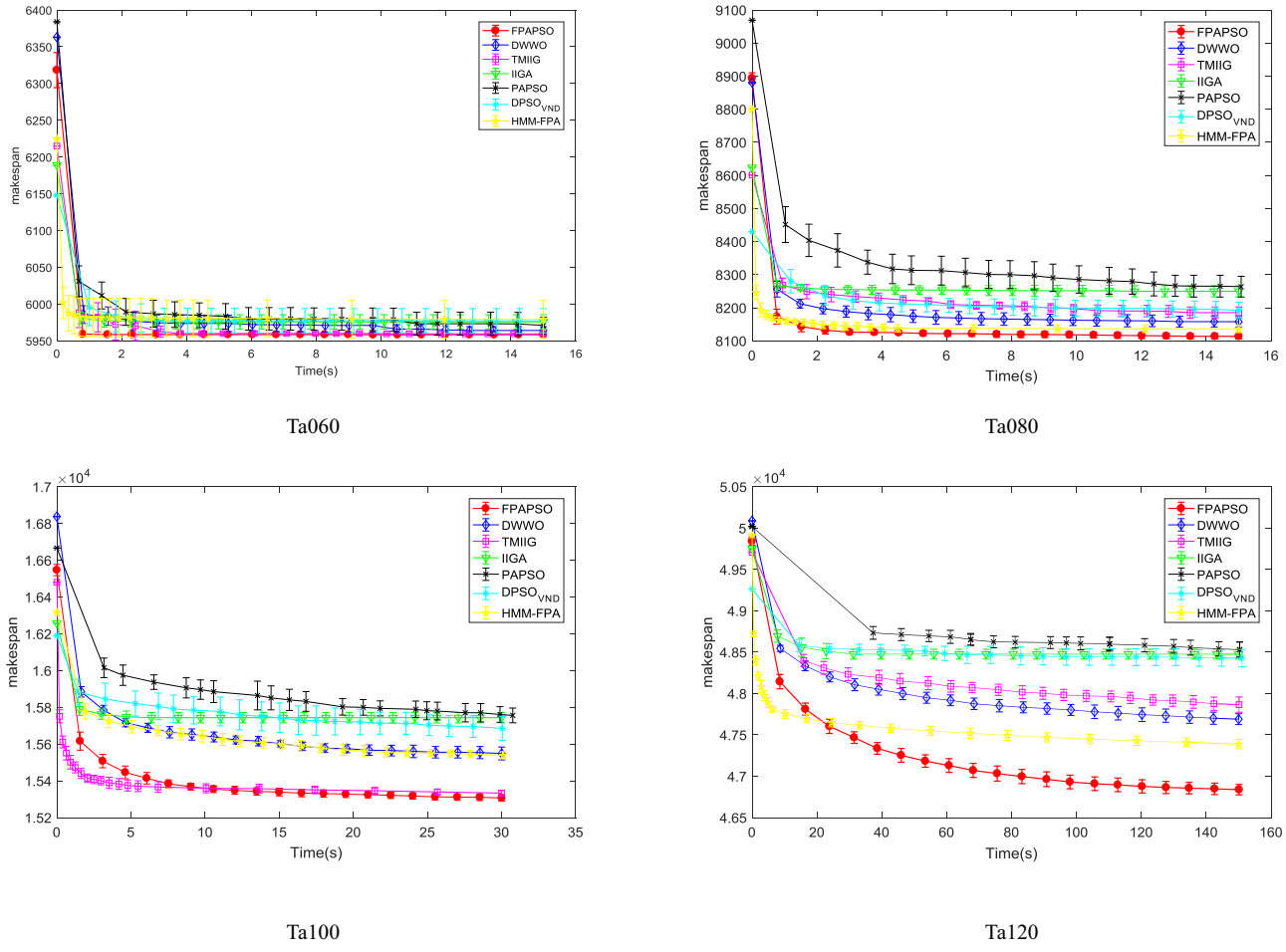

Fig. 2. The convergence curves on Taillard's benchmark.

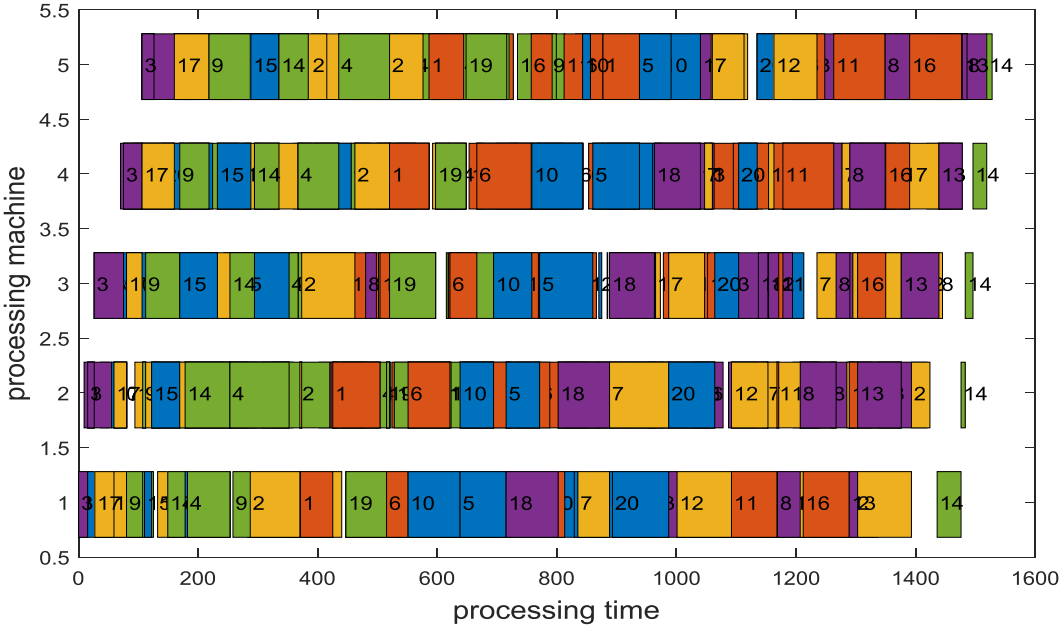

Fig. 3. The Gantt chart of the solution for Ta01.

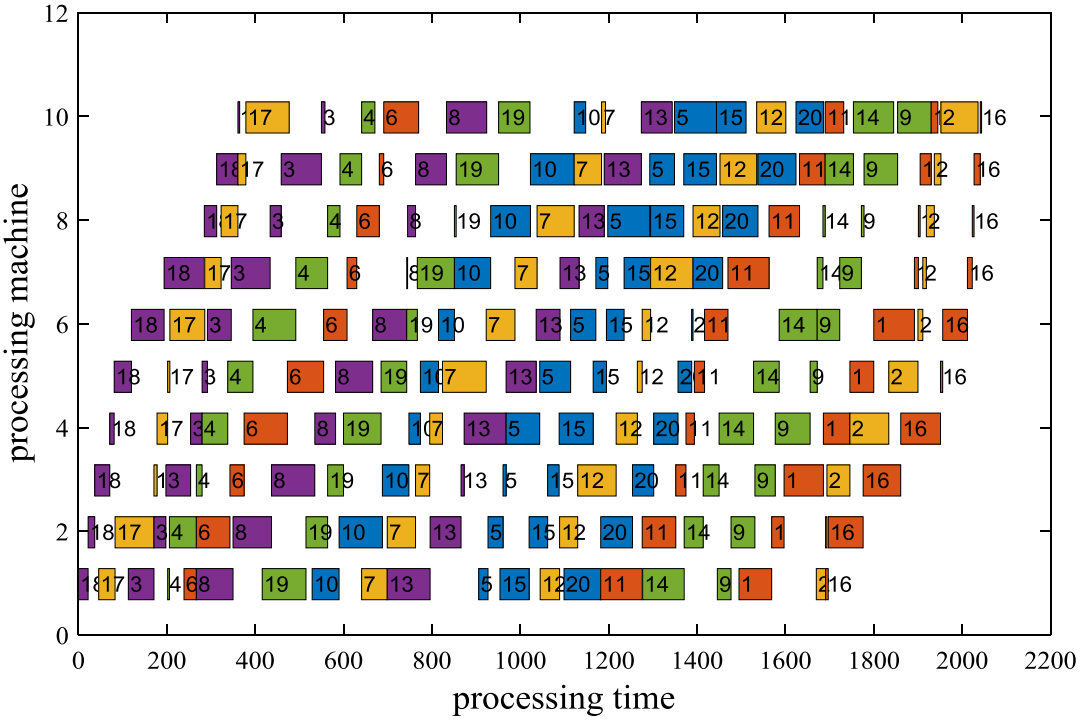

Fig. 4. The Gantt chart of the solution for Ta11.

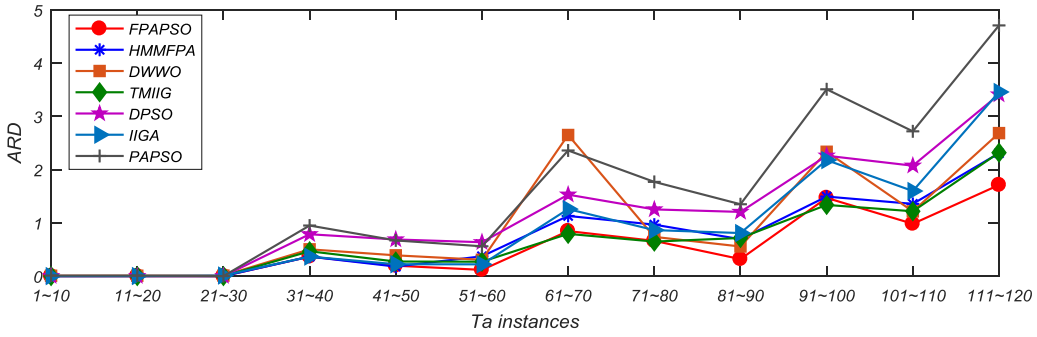

Fig. 5. The ARD of FPAPSO, PAPSO, HMMFPA DWWO, TMIIG, DPSO, IIGA for Ta instances.

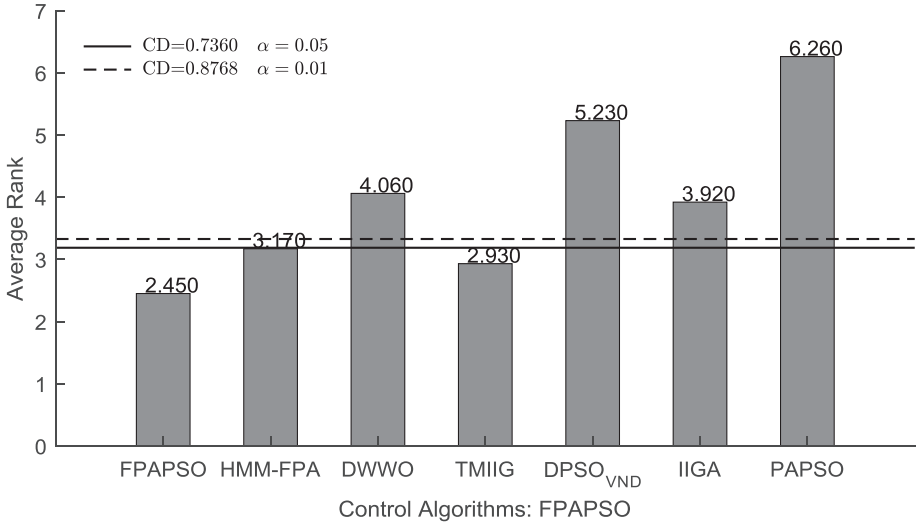

Fig. 6. Rankings obtained through the Friedman test and graphical representation.

Table 7
Results of the multiple-problem Wilcoxon's test at α=0.05 and α=0.01 significance level.

FPAPSO vs.	R+	R-	Z	p-value	$\alpha = 0.05$	$\alpha = 0.01$
PAPSO	4950.00	0.00	−8.64	5.69E−18	Yes	Yes
HMMFPA	3171.00	924.00	−4.52	6.00E−06	Yes	Yes
DWWO	4277.50	378.50	−7.12	1.04E−12	Yes	Yes
TMIIG	2845.50	1159.50	−3.45	5.62E−04	Yes	Yes
$DPSO_{VND}$	4095.00	0.00	−8.24	1.74E−16	Yes	Yes
IIGA	3928.50	257.50	−7.27	3.72E−13	Yes	Yes

[a] "Yes" indicates that statistically significant can be observed under the corresponding α.

Algorithm 1 Factorial encode process.

Input: Job permutation $\pi = \{J_1 J_2,..., J_{j-1}, J_j,..., J_n\}$
Start
  Initialize: $D_F = \{0, 1, ..., n-1\}$, $P = \{1, 2, ..., n\}$, $F = \{\}$, $N = 0$
  For $i = 1$: n
    $index = find(P[index] == J_i)$ // Find the first position index from left to right in P which makes $P[index]$ equal to J_i
    $F[i] = D_F[index]$
    remove $P[index]$ from P
    $N = N + F[i] \times D_F[n-i+1]!$
  End
End
Output: Natural number N

Algorithm 2 Factorial decode process.ab.

Inputs: Natural number: N, Number of jobs: n
Start
  Initialize: $PT = \{(n-1)!, (n-2)!, ..., 1!, 0\}$, $D_F = \{0, 1, ..., n-1\}$, $P = \{1, 2, ..., n\}$, $F = \{\}$, $\pi = \{\}$
  $index = find(PT[\text{index}] \le N)$ // Find the first position index from left to right in PT which makes $PT[index]$ not greater than N
  $F[1: index-1] = 0$
  While $N \neq 0$
    $F[index] = \text{quotient}(N/PT[index])$
    $N = remainder(N/PT[index])$
    $index = index + 1$
  End
  For $i = 1$: n
    index $= find(F, F[index] == D_F[i])$
    $\pi[i] = P[\text{index}]$
    remove $P[\text{index}]$ from P
  End
End
Output: Job permutation: π

Algorithm 3 Initialize permutations.

Inputs: A set of all jobs $P = \{J_P^1, J_P^2, \ldots, J_P^n\}$, the population size L
Start
  Randomly pick out L jobs $S = \{J_1 J_2,..., J_L\}$ from P
  For $j = 1$: L
    $J_{\pi_j}^1 = J_P^1$
    $J_{\pi_j}^2 = NN(J_{\pi_j}^1)$
    $S' = S - \{J_{\pi_j}^1, J_{\pi_j}^2\}$
    $\pi_j = [J_{\pi_j}^1, J_{\pi_j}^2] + NEH(S')$
    $\Pi_0^j = \pi_j$
  End
End
Output: The initial permutations: Π_0

Algorithm 4 VNS local search.

Inputs: A job permutation $\pi = \{J_1 J_2,..., J_{j-1}, J_j,..., J_n\}$, number of jobs: n
Start
  Initialize: $\eta = rand(1, n)$, $k = rand(1, n)$, $\eta \neq k$, $i = 0$
  $S = insert(\pi, \eta, k)$ // perturbation
  While $i < n(n-1)$
    $S' = SwapLocalSearch(S)$
    $S' = InseartLocalSearch(S')$
    If $fitness(S') \le fitness(S)$
      $S = S'$
      $i = 0$
    Else
      $i = i + 1$
    End
  End
  If $fitness(S) \le fitness(\pi)$
    $\pi = S$
  End
End
Output: Local optimum permutation π

cases. The statistical significance is observed in all the comparisons as the Wilcoxon's test with $\alpha = 0.05$ and $\alpha = 0.01$, which means that FPAPSO is significantly better than HMM-FPA, DWWO, TMIIG, $DPSO_{VND}$ and IIGA on solving $F_m|nwt|C_{max}$ problems with $\alpha = 0.05$ and $\alpha = 0.01$.

As to the above comparisons and discussion, there are the following conclusions. First, the PSO with factorial representation explores the search space of the Reeves instances and the Taillard's instances efficiently. Second, although the factorial representation method presents difficulties to exploit the best areas, the VNS local search method allows the PSO to exploit a promising area of the search space. Third, the population adaption allows the proposed algorithm to avoid being trapped into a local optimum since the

Algorithm 5 Population adaptation procedure.

Inputs: Population at the gth generation: $X_g = \{x_1, x_2,..., x_L\}$, distance of the population of last generation: D_{g-1}, distance unchanged times c, $pbest$ of current generation
Start
  Compute D_g using Eq. (13)
  If $D_g == D_{g-1}$
    $c = c + 1$
  Else
    $c = 0$
  End
  If $c \ge L$
    Get μ using Eq. (15)
    Get σ using Eq. (16)
    Regenerate population $X_g' = \{x_1', x_2', \ldots, x_L'\}$ using Eq. (14)
    Select an individual from X_g' randomly and replace it with $pbest$
    $c = 0$
  End
  $X_g = X_g'$
End
Outputs: Adapted population X_g, distance unchanged times c

Algorithm 6 Proposed FPAPSO algorithm.

Input: A set of all jobs $P = \{J_P^1, J_P^2, \ldots, J_P^n\}$
Start
  Initialize: L, MRT, c_1, c_2, $w_{\min}$, $w_{\max}$, $v_{\max}$, $c = 0$, $g = 1$
  $\Pi_0 = InitializePopulation(P, L)$
  Evaluate Π_0 to get the *pbest*, *gbest*, D_0
  $X_0 = FactoradicEncode(\Pi_0)$
  Generate feasible initial velocity V_1
  While $time_g < MRT$
    $X_g = X_{g-1} + V_g$
    $\Pi_g = FactoradicDecode(X_0, n)$
    Evaluate Π_g to update the *pbest*, *gbest*
    $VNS(pbest)$
    $X_g = FactoradicEncode(\Pi_g)$
    $[X_g, c] = PA(X_g, D_{g-1}, c)$
    Calculate feasible V_{g+1}
    $g = g + 1$
  End
End
Outputs: The optimal makespan *gbest* and corresponding permutation $FactoradicDecode(gbest)$

factorial represented landscapes are locally highly rugged. However, the FPAPSO takes a lot of extra run time in the encoding and decoding processes even though the computational complexity of these two methods are both O(n). The proposed FPAPSO is effective, efficient and robust for solving the NWFSP with the minimization of makespan criterion.

5. Conclusion and future research

In this paper, a factorial based particle swarm optimization with a population adaptation mechanism is proposed to solve the NWFSP. First, the factorial representation is employed as a novel coding method to transfer the permutation domain to an integer domain. Second, the NN+NEH method is introduced to generate potential permutations for PSO to start the search process. Third, the VNS local search method is introduced to exploit the promising area around the current best solution obtained by PSO. Finally, the population adaptation mechanism is designed to control the diversity of the population and avoid the particles being trapped into local optima. The computational results and comparisons based on the Reeve's and the Taillard's instances demonstrate the effectiveness, efficiency and robustness of the proposed FPAPSO.

The proposed FPAPSO algorithm also has the following limitation. The FPAPSO takes a lot of extra run time in the encoding and decoding processes as the existing generic computer architecture is difficult to deal with the large integer operations brought by a factorial.

Further work is divided into following directions. First, it is necessary to alter the factorial representation to reduce the complexity of the algorithm. Second, it is possible to employ the factorial representation to encode various evolutionary algorithms, which includes a differential evolution algorithm and biogeography-based optimization, to deal with the NWFSP. Finally, the FPAPSO algorithm could be applied for solving other complex scheduling problems, such as the flexible flow shop scheduling problems, the job shop scheduling problems and the hybrid flow shop scheduling problems in the literature.

Credit authorship contribution statement

Fuqing Zhao: Funding acquisition, Investigation, Supervision, Writing - review & editing. **Shuo Qin:** Investigation, Software, Writing - original draft. **Guoqiang Yang:** Conceptualization, Formal analysis. **Weimin Ma:** Methodology, Resources. **Chuck Zhang:** Project administration, Writing - review & editing. **Houbin Song:** Visualization.

Acknowledgment

This work was financially supported by the National Natural Science Foundation of China under grant numbers 61663023. It was also supported by the Key Research Programs of Science and Technology Commission Foundation of Gansu Province (2017GS10817), Lanzhou Science Bureau project (2018-rc-98), Public Welfare Project of Zhejiang Natural Science Foundation(LGJ19E050001), Wenzhou Public Welfare Science and Technology project (G20170016), respectively.

References

Akhshabi, M., Tavakkoli-Moghaddam, R., & Rahnamay-Roodposhti, F. (2014). A hybrid particle swarm optimization algorithm for a no-wait flow shop scheduling problem with the total flow time. *International Journal of Advanced Manufacturing Technology, 70*(5–8), 1181–1188.
Chen, Y., Li, L., Xiao, J., Yang, Y., Liang, J., & Tao, L. (2018). Particle swarm optimizer with crossover operation. *Engineering Applications of Artificial Intelligence, 70*, 159–169.
Corus, D., Dang, D.-C., Eremeev, A. V., & Lehre, P. K. (2014). Level-Based Analysis of Genetic Algorithms and Other Search Processes. In T. BartzBeielstein, J. Branke, B. Filipic, & J. Smith (Eds.), *Parallel Problem Solving from Nature-PPSN XIII 8672* (pp. 912–921) (Eds.).
Ding, J. Y., Song, S., Gupta, J. N. D., Rui, Z., Chiong, R., & Cheng, W. (2015a). An improved iterated greedy algorithm with a Tabu-based reconstruction strategy for the no-wait flowshop scheduling problem. *Applied Soft Computing, 30*, 604–613.
Ding, J.-Y., Song, S., Zhang, R., Zhou, S., & Wu, C., A Novel Block-shifting Simulated Annealing Algorithm for the No-wait Flowshop Scheduling Problem 2015b.
Eberhart, R., & Kennedy, J (1995). A new optimizer using particle swarm theory. *International Symposium on MICRO Machine and Human Science* (pp. 39–43).
Fink and Voß. (2003). Solving the continuous flow-shop scheduling problem by metaheuristics. *European Journal of Operational Research, 151*(2), 400–414.
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 Real Parameter Optimization. *Journal of Heuristics, 15*(6), 617–644.
Garey, M. R., Johnson, D. S., & Sethi, R. (1976). The Complexity of Flowshop and Jobshop Scheduling. *Mathematics of Operations Research, 1*(2), 117–129.
Knuth, D. E. (1998). Art of Computer Programming, Volume 3: Sorting and Searching (2nd Edition).
Laha, D., & Chakraborty, U. K. (2009). A constructive heuristic for minimizing makespan in no-wait flow shop scheduling. *International Journal of Advanced Manufacturing Technology, 41*(1–2), 97–109.
Laisant, C.-A. (1888). Sur la numération factorielle, application aux permutations. *Bull.soc.math.france, 2*, 176–183.
Lin, S.-W., & Ying, K.-C. (2016). Optimization of makespan for no-wait flowshop scheduling problems using efficient matheuristics. *Omega-International Journal of Management Science, 64*, 115–125.
Liu, B., Wang, L., & Jin, Y.-H. (2007). An effective hybrid particle swarm optimization for no-wait flow shop scheduling. *International Journal of Advanced Manufacturing Technology, 31*(9–10), 1001–1011.
Mladenović, N., & Hansen, P. (2008). Variable neighborhood search. *European Journal of Operational Research, 191*, 593–595.
Nawaz, M., Enscore, E. E., Jr, & Ham, I. (1983). A heuristic algorithm for the m -machine, n -job flow-shop sequencing problem. *Omega, 11*(1), 91–95.
Pan, Q.-K., Tasgetiren, M. F., & Liang, Y.-C. (2008a). A discrete particle swarm optimization algorithm for the no-wait flowshop scheduling problem. *Computers & Operations Research, 35*(9), 2807–2839.
Pan, Q.-K., Wang, L., & Zhao, B.-H. (2008b). An improved iterated greedy algorithm for the no-wait flow shop scheduling problem with makespan criterion. *International Journal of Advanced Manufacturing Technology, 38*(7), 778–786.
Qu, C., Fu, Y., Yi, Z., & Tan, J. (2018). Solutions to No-Wait Flow Shop Scheduling Problem Using the Flower Pollination Algorithm Based on the Hormone Modulation Mechanism. *Complexity, 2018*, 18.
Reeves, C. R. (1995). A genetic algorithm for flowshop sequencing. *Computers & Operations Research, 22*(1), 5–13.
Ribas, I., Companys, R., & Tort-Martorell, X. (2017). Efficient heuristics for the parallel blocking flow shop scheduling problem. *Expert Systems with Applications, 74*, 41–54.
Shao, W., Pi, D., & Shao, Z. (2017a). An extended teaching-learning based optimization algorithm for solving no-wait flow shop scheduling problem. *Applied Soft Computing, 61*, 193–210.
Shao, W., Pi, D., & Shao, Z. (2017b). Memetic algorithm with node and edge histogram for no-idle flow shop scheduling problem to minimize the makespan criterion. *Applied Soft Computing, 54*, 164–182.
Taillard, E. (1993). Benchmarks for basic scheduling problems. *Eur.j.oper.res, 64*(2), 278–285.
Ye, H., Li, W., & Abedini, A. (2017). An improved heuristic for no-wait flow shop to minimize makespan. *Journal of Manufacturing Systems, 44*, 273–279.

Ye, H., Li, W., & Miao, E (2016). An effective heuristic for no-wait flow shop production to minimize makespan. *Journal of Manufacturing Systems, 40*, 2–7.
Yu, J.-M., Huang, R., & Lee, D.-H. (2017). Iterative algorithms for batching and scheduling to minimise the total job tardiness in two-stage hybrid flow shops. *International Journal of Production Research, 55*(11), 1–17.
Zar, J. H. (1999). *Biostatistical analysis*. Prentice Hall.
Zhao, F., Chen, Z., Wang, J., & Zhang, C. (2017a). An improved MOEA/D for multi-objective job shop scheduling problem. *International Journal of Computer Integrated Manufacturing, 30*(6), 616–640.
Zhao, F., Liu, Y., Zhang, Y., Ma, W., & Zhang, C. (2017b). A hybrid harmony search algorithm with efficient job sequence scheme and variable neighborhood search for the permutation flow shop scheduling problems. *Engineering Applications of Artificial Intelligence, 65*, 178–199.
Zhao, F., Liu, H., Zhang, Y., Ma, W., & Zhang, C. (2018a). A discrete Water Wave Optimization algorithm for no-wait flow shop scheduling problem. *Expert Systems with Applications, 91*, 347–363.
Zhao, F., Liu, Y., Zhang, C., & Wang, J. (2015a). A self-adaptive harmony PSO search algorithm and its performance analysis. *Expert Systems with Applications An International Journal, 42*(21), 7436–7455.
Zhao, F., Qin, S., Zhang, Y., Ma, W., Zhang, C., & Song, H. (2019). A two-stage differential biogeography-based optimization algorithm and its performance analysis. *Expert Systems with Applications, 115*, 329–345.
Zhao, F., Shao, Z., Wang, J., & Zhang, C. (2016). A hybrid differential evolution and estimation of distribution algorithm based on neighbourhood search for job shop scheduling problems. *International Journal of Production Research, 54*(4), 1–22.
Zhao, F., Xue, F., Zhang, Y., Ma, W., Zhang, C., & Song, H. (2018b). A hybrid algorithm based on self-adaptive gravitational search algorithm and differential evolution. *Expert Systems with Applications, 113*, 515–530.
Zhao, F., Zhang, J., Zhang, C., & Wang, J. (2015b). An improved shuffled complex evolution algorithm with sequence mapping mechanism for job shop scheduling problems. *Expert Systems with Applications, 42*(8), 3953–3966.